VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

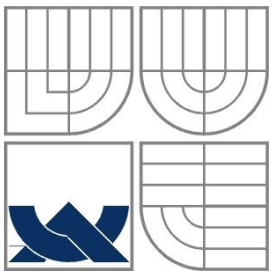
BIGDATA ŘEŠENÍ PRO ZPRACOVÁNÍ ROZSÁHLÝCH DAT ZE SÍŤOVÝCH TOKŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

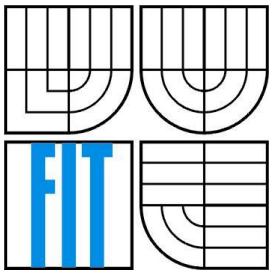
AUTOR PRÁCE
AUTHOR

Bc. MILOSLAV MELKES

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

BIGDATA ŘEŠENÍ PRO ZPRACOVÁNÍ ROZSÁHLÝCH DAT ZE SÍŤOVÝCH TOKŮ

BIGDATA APPROACH TO MANAGEMENT OF LARGE NETFLOW DATASETS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. MILOSLAV MELKES

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. ONDŘEJ RYŠAVÝ, Ph.D.

BRNO 2014

Abstrakt

Tato diplomová práce se zaměřuje na problematiku distribuovaného zpracování velkých dat ze síťové komunikace. Začíná analýzou síťové komunikace založené na modelu TCP/IP, se zaměřením na datové jednotky na jednotlivých vrstvách, které je nutno při analýze síťových dat zpracovávat. Z hlediska vlastního zpracování rozsáhlých dat je objasněn výpočetní model MapReduce, architektura technologie Apache Hadoop a jejich možné využití pro zpracování síťových toků na clusteru počítačů. Druhá část práce se zbývá návrhem a následnou implementací aplikace pro zpracování síťových toků ze zachycené síťové komunikace. V této části jsou rozebrány klíčové a problematické části z implementace. Celá práce je poté zakončena srovnáním s dostupnými nástroji pro síťovou analýzu a vyhodnocením sady testů, které potvrdili lineární růst zrychlení.

Abstract

This master‘s thesis focuses on distributed processing of big data from network communication. It begins with exploring network communication based on TCP/IP model with focus on data units on each layer, which is necessary to process during analyzation. In terms of the actual processing of big data is described programming model MapReduce, architecture of Apache Hadoop technology and it‘s usage for processing network flows on computer cluster. Second part of this thesis deals with design and following implementation of the application for processing network flows from network communication. In this part are discussed main and problematic parts from the actual implementation. After that this thesis ends with a comparison with available applications for network analysis and evaluation set of tests which confirmed linear growth of acceleration.

Klíčová slova

TCP/IP, libpcap, WinPcap, Netflow, PCAP, Wireshark, Big Data, MapReduce, Apache Hadoop, HDFS, MongoDB, síťový tok

Keywords

TCP/IP, libpcap, WinPcap, Netflow, PCAP, Wireshark, Big Data, MapReduce, Apache Hadoop, HDFS, MongoDB, network flow

Citace

Miloslav Melkes: BigData řešení pro zpracování rozsáhlých dat ze síťových toků, diplomová práce, Brno, FIT VUT v Brně, 2014

BigData řešení pro zpracování rozsáhlých dat ze síťových toků

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Ondřeje Ryšavého, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Miloslav Melkes
27. května 2014

Poděkování

Rád bych chtěl tímto poděkovat vedoucímu mé diplomové práce Ing. Ondřeji Ryšavému, Ph.D. za ochozu a cenné připomínky, které mně pomohli při tvorbě této práce.

Obsah

1 Úvod

Síťový provoz v dnešní době přestavuje obrovské množství. Chceme-li tyto data analyzovat, tak se jedná o náročnou operaci. Za prvé musíme být schopni tyto data někde uložit před zpracováním a mít dostatečnou volnou úložnou kapacity pro uložení výstupu. Za druhé potřebujeme tyto data analyzovat v rozumném časem, abychom mohli výsledky analyzovat.

Pokud se jedná přímo o zachycené pakety přenášené po síti, tak jde opravu v delším horizontu o velké množství dat, které roste se zvyšujícím se provozem na síti. Proto se nabízí využít distribuované zpracování a uložení těchto dat, což umožní paralelní zpracování na jednotlivých uzlech distribuovaného systému, na němž zpracování probíhá. Nejlépe použít takové mechanismy, aby bylo vytvořené řešení dobře škálovatelné.

Cílem tohoto projektu je prozkoumat možnosti distribuovaného zpracování zachycené síťové komunikace. Ze získaných poznatků následně navrhnout a implementovat aplikaci, která bude využívat dostupných nástrojů.

První část práce je zaměřena na prozkoumání domény, která se musí řešit při návrhu a implementaci aplikace provádějící distribuované zpracování síťových dat. Zahrnuje problematiku síťové komunikace, zachycených síťových dat a vlastního distribuovaného zpracování. Druhá část práce zahrnuje návrh a implementaci aplikace pro distribuované zpracování zachycených síťových dat se zhodnocením dosažených výsledků.

Následující kapitola se zabývá problematikou síťové komunikace. Obsahuje vysvětlení sítové komunikace založené na vrstvovém modelu TCP/IP. Jsou v ní popsány úlohy jednotlivých vrstev z tohoto modelu. Zejména je kladen důraz na datové jednotky na jednotlivých vrstvách a jejich vzájemné zapouzdření. Zde jsou důležité informace odesílateli, příjemci a předmětu komunikace. Dále je v této kapitole popsán způsob zachytávání síťové komunikace. Jedná se dostupný software pro analýzu a zachytávání síťového provozu včetně knihoven, na kterých je založen. V neposlední řadě je popsán formát pro ukládání zachycených síťových dat, se kterým je počítáno v návrhu aplikace.

Třetí kapitola se soustředí na techniky a prostředky pro distribuované zpracování velkých dat. Objasňuje pojem „velkých dat“ a jejich specifik, na které je brát při zpracování zřetel. Dále je v této kapitole popsán výpočetní model MapReduce pro zpracování velkých dat s ukázkou principu jeho použití. Na něj navazuje framework Apache Hadoop využívající výpočetní model MapReduce pro zpracování velkých dat. Obsažen je popis jeho architektonických částí jak pro ukládání dat, tak pro výpočetní část včetně možností jeho využití.

Návrh možného řešení pro zpracování síťový dat je rozdělen do dvou částí. První se zabývá zpracováním síťových dat s využitím Apache Hadoop a programovacího modelu MapReduce. Vysvětlení zahrnuje princip zpracování síťových data s využitím tohoto modelu, který je možné implementovat na této platformě. Druhá část je zaměřena na návrh pro dotazování se nad výsledky zpracovaných síťových dat.

Na implementací výsledné aplikace je zaměřena pátá kapitola. V té jsou uvedeny nejprve použité technologie, které byly využity při vývoji aplikace. Navržené struktura celého řešení a jeho rozdělení do logických část. Hlavně ale způsob zpracování zachycené síťové komunikace přes programovací model MapReduce, Jsou uvedeny jednotlivé fáze zpracování včetně podrobného rozboru řešení, které bylo implementováno ve výsledné aplikaci.

Šestá kapitola se zabývá zhodnocení implementovaného řešení. To je založeno na testech provedených ve virtuálním prostředí. U výsledků testů nejde ani tak o získané absolutní hodnoty, ale spíše o relativní rozdíly mezi získanými hodnotami při různých konfiguracích.

Poté následuje předposlední kapitola zaměřená na srovnání s dostupnými nástroji a možná rozšíření aplikace. Z hlediska srovnání je zaměřena na porovnání s nástroji pro zpracování a analýzu síťové komunikace. Ve druhé části je potom zaměřena na možné rozšíření aplikace v podobě podpory většího množství aplikační protokolů a dalších možností analýzy síťových dat.

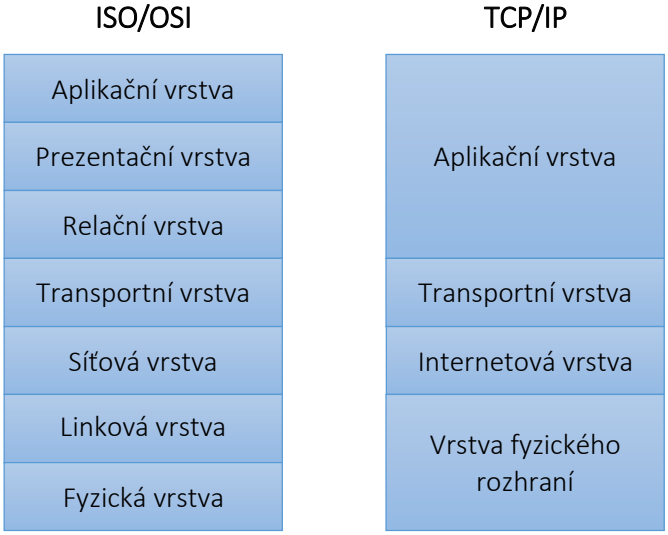
2 Analýza síťové komunikace

2.1 Síťová komunikace

V tomto projektu se zaměříme na zpracování síťového provozu z modelu TCP/IP, který vychází z referenčního ISO/OSI modelu. Vzhledem ke složitosti síťové komunikace se referenční model skládá ze sedmi vrstev, které jsou definovány službami a odpovídajícími protokoly. Tyto vrstvy popisují funkce pro přenos dat mezi procesy stejné vrstvy. Konkrétní vrstva se přitom využívá při komunikaci služeb nižších vrstev, bez znalosti implementace či chování této nižší vrstvy. Z jejího pohledu je přenos po síti pouze záležitostí nejbližší vrstvy.

2.1.1 Architektura TCP/IP

Model TCP/IP se oproti referenčnímu modelu ISO/OSI skládá pouze ze čtyř vrstev a referenční model tedy není zcela implementován, protože je velice komplexní, což by způsobilo velice složitou implementaci. Vrstvy relační, prezentační a aplikační z referenčního modelu jsou spojeny do jedné aplikační vrstvy v modelu TCP/IP, jak ukazuje obrázek Obrázek *2.1*. Dále jsou spojeny vrstva fyzická a linková do větší vrstvy fyzického rozhraní. Funkce jednotlivých vrstev modelu TCP/IP následují.



Obrázek 2.1: Referenční síťový model ISO/OSI a model TCP/IP.

Vrstva fyzického rozhraní
Nejnižší vrstva, která zajišťuje přístup k fyzickému médiu a zabalování datagramů do rámců. Popisuje standardy pro přístup na fyzické médium a elektrické signály. Je implementována v síťové kartě a jejím ovladači.

Internetová vrstva
Zajišťuje logické spojení mezi počítači. Pracuje s protokolem IP, podle kterého se směrují datagramy na cílové místo dané síťovou IP adresou. Doručení dat ovšem není spolehlivé, protože tato vrstva se snaží data doručit s největším úsilím tzv. best-effort delivery. Pokud tedy nastane ztráta dat během přenosu, tak je o tom odesílatel informován a musí sám zajistit opětovné přenesení dat. Na této vrstvě se kromě protokolu IP používají také protokoly ARP (Address Resolution Protocol) a RARP (Reverse ARP) pro mapování IP adres na MAC adresy. Dále protokol ICMP (Internet Control Message Protocol)

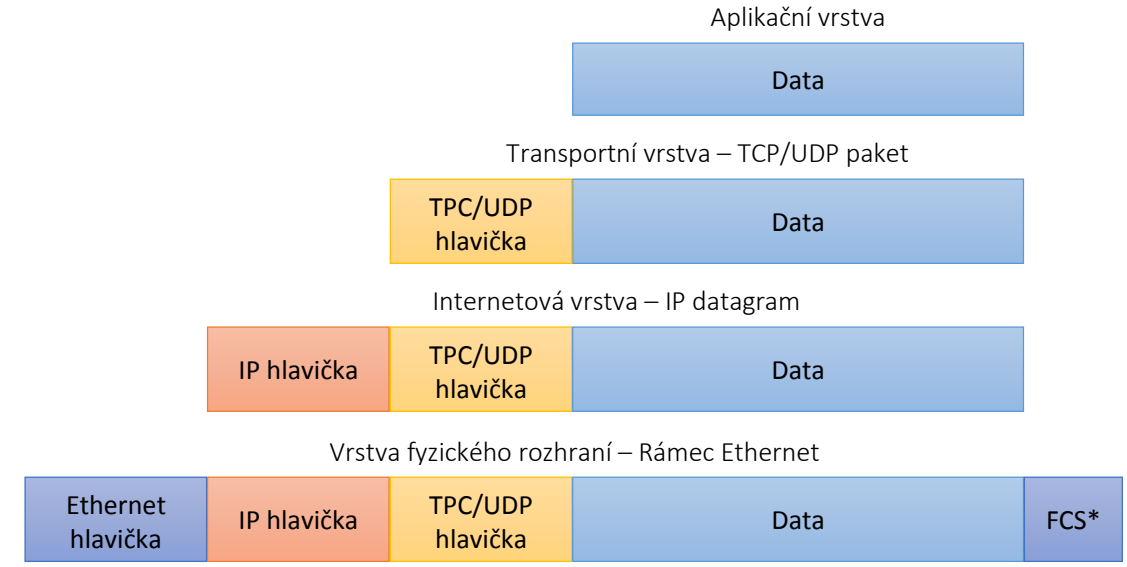
pro řízení toku a detekci nedosažitelných cílů a protokol IGMP (Internet Group Management Protocol) pro přihlašování do multicastových skupin. Vrstva je implementována v modulu operačního systému tzv. IP stack.

Transportní vrstva
Tato vrstva obstarává logické spojení mezi aplikacemi či procesy, které vytvářejí nebo zpracovávají aplikační data. Data z aplikační vrstvy se na této vrstvě rozdělují do tzv. paketů a přidává se k nim hlavička paketu dle použitého transportního protokolu. Mezi základní protokoly řadíme TCP (Transmission Control Protocol) pro spolehlivý přenos dat a UDP (User Datagram Protokol) pro rychlý, ale nespolehlivý přenos dat. Tato vrstva je implementována v modulu operačního systému stejně jako vrstva internetová.

Aplikační vrstva
Komunikace na nejvyšší úrovni mezi procesy/aplikacemi, které běží na počítači. Jelikož dle ISO/OSI je rozdělena tato vrstva na tři různé , tak tato jediná v TCP/ICP modelu řeší také prezentaci dat a správu sezení. Prezentace dat zahrnuje kódování dat pro přenos a převod z tohoto kódování zpět. Udržování a vytváření sezení tzv. session, které představuje kontext komunikace. Implementována přímo v aplikaci nebo jako systémová služba.

2.1.2 Zapouzdření dat

Zapouzdření dat v síti TCP/IP zobrazuje obrázek *Obrázek 2.2*. Každá vrstva z TCP/IP modelu, mimo vrstvu aplikační, přidává k datům z vyšší vrstvy svoji hlavičku. Odeslání a příjem funguje následovně. Na počátku odesílání máme data vytvořená aplikační vrstvou. Tato data jsou předána transportní vrstvě, kde se použije transportní protokol což je nejčastěji TCP nebo UDP. K datům se tedy přidá TCP nebo UDP hlavička, kde jsou jako identifikátory aplikací použity zdrojový a cílový port. Takto vytvořený paket je předán internetové vrstvě, kde se přidá IP hlavička obsahující zdrojovou a cílovou IP adresu a další hodnoty a je vytvořen IP datagram. IP datagram dále zpracovává vrstva síťového rozhraní, která přidává hlavičku se zdrojovou a cílovou MAC adresou a je vytvořen rámec, který je odesílán na médium. Při příjmu na cílovém počítači se postupuje opačným způsobem. Od vrstvy fyzického rozhraní se postupně kontrolují a rozbalují datové jednotky až jsou aplikační data předána cíli.

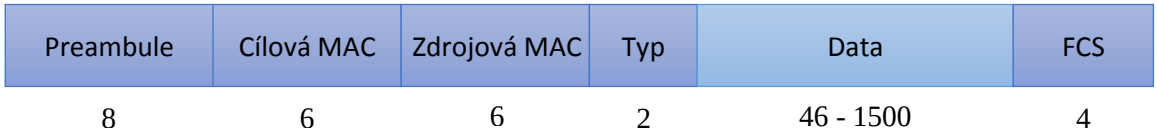


*Frame check sequence

Obrázek 2.2: Zapouzdření dat na jednotlivých vrstvách TCP/IP modelu.

2.1.3 Rámec Ethernet

Data na vrstvě fyzického rozhraní obsahují ethernet hlavičku, pole frame check sequence a data z vyšší vrstvy. Všechny tyto části dohromady nazýváme Ethernet rámec. Na základě tohoto rámce dokážeme jednoznačně identifikovat síťové rozhraní díky obsahu ethernet hlavičky. Tato hlavička totiž obsahuje fyzické MAC (Media Access Control) adresy síťových rozhraní odesílatele a příjemce. Používá se díky tomu pro doručování dat cílovému rozhraní v lokálních sítích LAN. Přesněji řečeno rozhraní rozpozná svoji MAC adresu a rámec přijme. Prvky, které pracují na L2 vrstvě ISO/OSI modelu se řídí informacemi v ethernet hlavičce tedy např. L2 přepínač.



Obrázek 2.3: Struktura rámce Ethernet II

2.1.4 IP datagram

Název datagram je označení pro blok dat odesílaných od zdrojového k cílovému hostiteli, kteří jsou identifikováni pomocí adres pevné délky. V případě protokolu IP jsou to IP adresy IPv4 nebo IPv6 dle verze použitého protokolu. Úkolem protokolu IP je přepravit data z vyšší vrstvy od zdroje k cíli. Protokol IP tuto činnost provádí bez záruky tzn. není garantováno doručení, zachování pořadí ani vynechání duplicit. Tyto záruky si musí zajistiti protokol na vyšší vrstvě. Stejně jako kontrolu integrity dat, protože IP hlavička obsahuje pouze kontrolní součet hlavičky. S tímto protokolem pracují síťová zařízení pracující na L3 vrstvě ISO/OSI modelu (např. směrovač).

Maximální velikost IP datagramu je omezena na 1500 bajtů (hodnota MTU[1]), jak je vidět na obrázku *Obrázek 2.3*. Velikost IP datagramu může být ovšem větší. Proto je nutné provádět tzv. fragmentaci, kdy je větší IP datagram rozdělen na několik menších. Ovšem nevýhodou této metody je, že může způsobovat nadměrné přenosy dat. Pokud dojde ke ztrátě i jediného fragmentu, tak pokud si protokol na vyšší vrstvě vyžádá znovu zaslání, tak se zasílá celý původní IP datagram.

2.1.4.1 IPv4 hlavička

Velice rozšířený protokol internetové vrstvy, který používá adresy délky 32 bitů. V současnosti se již více nerozšiřuje, protože došlo k rozdělení posledních bloků adres a tím pádem k jejich vyčerpání.

Bajty	1		2	3	4
0-4	Verze	IHL	Typ služby	Celková délka	
5-8	Identifikace			Příznaky	Offset fragmentu
9-12	Time To Live		Protokol	Kontrolní součet hlavičky	
13-16	Zdrojová adresa				
17-20	Cílová adresa				
20-(IHL*4)-1	Rozšiřující informace				Zarovnání

Tabulka 2.1: Struktura IPv4 hlavičky

[1] Maximum Transmission Unit

Legenda tabulky Tabulka *2.1*:

- IHL – Velikost IPv4 hlavičky ve slovech délky 32 bitů.
- Type of service (TOS) – Indikace abstraktních parametrů o žádoucí kvalitě služby.
- Celková délka – Velikost IP datagramu v bajtech.
- Identifikátor – Odesílatel přidělí každému paketu jednoznačný identifikátor. Fragmentované datagramy mají stejný identifikátor.
- Příznaky – Určeny pro řízení fragmentace.
- Offset fragmentu – Pozice, na které začíná fragment v původním datagramu. Jeho jednotkou je osm oktetů.
- Time To Live (TTL) – Maximální čas platnosti datagram. Jeho hodnota je snížena o 1 při zpracování datagramu zařízením sítě. Pokud dosáhne hodnoty nula, tak musí být zahozen. Tím je zajištěno, že se datagram nezacyklí v síti např. při chybě ve směrovacích tabulkách a po určitém počtu směrování se paket zahodí. Při každé změně této hodnoty je nutné přepočítat a zapsat nový kontrolní součet hlavičky.
- Protokol – Určuje protokol na vyšší vrstvě. Tedy kterému protokolu se mají data předat.
- Kontrolní součet hlavičky – Používá se k ověření integrity IP hlavičky. Pokud nesouhlasí, tak se datagram zahodí

2.1.4.2 IPv6 hlavička

Protokol IPv6 je nástupce staršího protokolu IPv4. Přináší výrazné rozšíření adresového prostoru díky použití adres délky 128 bitů. Dochází také ke zjednodušení IPv6 hlavičky oproti IPv4 hlavičce, protože obsahuje pouze nejnutnější informace a díky tomu je možné její rychlejší zpracování. Další rozšiřující hlavičky jsou případně zřetězeny za sebou, kdy vždy typ další hlavičky udává položka „Další hlavička“. Také se odstranil se kontrolní součet hlavičky a je jej nutné počítat na každém směrovači. Mezi další výhody patří menší velikost směrovací tabulek, bez stavová konfigurace adres a eliminace všesměrového vysílání.

Bajty	1	2	3	4
0-4	Verze	Třída provozu	Značka toku	
5-8	Délka dat		Další hlavička	Max. skoků
9 – 24	Zdrojová adresa			
25 – 40	Cílová adresa			

Tabulka 2.2: Struktura IPv6 hlavičky

Legenda tabulky Tabulka *2.2*:

- Třída provozu – Pro rozlišení různě prioritního provozu.
- Značka toku – Pro optimalizaci směrování sledu datagramů, které tvoří jeden logický celek.
- Délka dat – Velikost dat v oktetech následujících za IPv6 hlavičkou.
- Další hlavička – Identifikuje typ hlavičky následující bezprostředně za touto.
- Max. skoků – Hodnota snižovaná o 1 každým uzlem sítě, který jej směřuje k cíli. Pokud dosáhne hodnoty nula, tak je datagram zahozen.

2.1.5 TCP/UDP paket

Protokol IP dopravuje IP datagram mezi počítači a jako identifikátor používá IP adresy. Protokoly transportní vrstvy přepravují data mezi konkrétními procesy/aplikacemi. K identifikaci procesu/aplikace používají 16bitová čísla portů. Tyto čísla portů jsou rozdělena do tří skupin [1]

- všeobecně známé porty (well known ports) v rozsahu 0 - 1023,
- registrované porty v rozsahu 1024 – 49151,
- dynamické a privátní porty v rozsahu 49152 – 65535.

Všeobecně známé a registrované porty přiděluje organizace IANA[1] [2] a používají je zejména serverové aplikace, aby klienti věděly, na které porty se připojovat.

2.1.5.1 Protokol TCP

Protokol TCP (Transmission Control Protocol) se používá pro služby se spojením a toto spojení se vytvoří před začátkem přenosu dat a po jeho skončení se ukončí. Zasílá data z aplikační vrstvy jako proud dat, který je tvořen z číslovaných paketů zasílaných v určeném pořadí k cíli. Zajišťuje spolehlivý přenos dat. K tomu má implementováno řízení toku, přijetí paketů ve správném pořadí, jejich potvrzování a řízení zahlcení. Odesílatel i příjemce pracuje s tzv. posuvným okénkem (sliding window), jehož velikost se v průběhu přenosu mění. Jeho velikost udává maximální možní počet nepotvrzených paketů. Hlavička TCP protokolu je v tabulce *Tabulka 2.4*.

Bity	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Zdrojový port																Cílový port															
32	Číslo sekvence																															
64	Potvrzený paket																															
96	offset dat				Rezervováno				URG	ACK	PSH	RST	SYN	FIN			Velikost okénka															
128	Kontrolní součet																Pozice urgentních dat															
160	Rozšiřující informace																														Zarovnání	
*[2]	Data																															

Tabulka 2.3: Struktura TCP paketu

2.1.5.2 Protokol UDP

Protokol UDP (User Datagram Protocol) na rozdíl od protokolu TCP nevytváří spojení před zahájením komunikace, a proto se používá pro nespojované služby. Není zde tedy implementováno potvrzování paketů a jedná se o nespolehlivý přenos dat. Odesílatel vytvoří a odešle UDP paket k příjemci, a pokud se po cestě paket ztratí, tak se o to protokol UDP nestará. O znovu zaslání se musí postarat aplikační protokol [2]. Díky nepotvrzování přijatých paketů a nevytváření spojení si hlavička UDP vystačí pouze se čtyřmi 16 bitovými položkami, což je podstatně méně než u protokolu TCP. Ovšem při jeho použití přicházíme o spolehlivost přenosu a zachování pořadí paketů. Z těchto důvodů se protokol UDP používá zejména pro streamování videa, kde potřebujeme přenést velké množství dat v krátkém čase a výpadek nějakého paketu nám až tak nevadí.

[1] http://www.iana.org/
[2] Offset dat * 8

Bajty	0	1	2	3
0	Zdrojový port		Cílový port	
4	Velikost UDP paketu		Kontrolní součet	
*[1]	Data			

Tabulka 2.4: Struktura UDP paketu

2.1.6 Síťový tok

Komunikace v počítačových sítích se typicky skládá ze síťových toků. Za síťový tok považujeme posloupnost paketů mající společné vlastnosti odvozené z obsahu paketu. Může se tedy např. jednat o pakety se stejnou zdrojovou a cílovou IP adresou, stejným zdrojovým a cílovým portem a stejným protokolem na transportní vrstvě. Za síťový tok můžeme např. považovat pakety nesoucí části video streamu ze serveru ke klientovi.

2.2 Zachytávání síťového provozu

Zachytávání síťového provozu je proces síťové analýzy s cílem získat informace o provozu na síti. Toto zachytávání provádí program zvaný sniffer [3], který naslouchá na daném rozhraní a čte každý rámec, který je určený pro toto rozhraní. V případě, že tento program umí přepnout síťové rozhraní do promiskuitního režimu, tak dokáže zachytávat veškerou komunikaci na médiu. To představuje velkou bezpečnostní hrozbu, protože sniffer může být spuštěn prakticky na kterémkoli počítači a jeho uživatel o tom nemusí vůbec vědět. V takovém případě jsou sniffery obtížně detekovatelné.

Data jsou zachytávána na L2 vrstvě ISO/OSI modelu. V případě ethernetu se jedná ethernet rámce popsané v kapitole 2.1.3. Máme tedy zapouzdřena v odchycených datech veškerá data z vyšších vrstev a můžeme využít různé úrovně analýzy, za pomocí dostupných nástrojů. Další možností zpracování je ukládat si zachycené pakety do souborů ve formátech k tomu určených a využít je k dalšímu zpracování ve vlastních aplikacích.

2.2.1 Knihovny libpcap a WinPcap

Libpcap[2] a WinPcap[3] jsou knihovny napsané v jazyce C, které jsou určeny pro použití v software na zachytávání a analýzu síťového provozu a jiných síťových aplikacích. Knihovna libpcap byla vytvořena z programu tcpdump, který je určen pro analýzu síťového provozu. Vznikla takže, že se z něj vyextrahovala nízko úrovňová implementace funkcí pro zachytávání paketů, čtení souborů se zachycenými daty a ukládání zachycených dat do souborů a vytvořila se samostatná knihovna. Verze WinPcap představuje port knihovny libpcap do prostředí Windows, ve kterém využívá NDIS (Network Driver Interface Specification) rozhraní pro čtení paketů přímo ze síťového rozhraní. Používá přitom nízko úrovňové programové rozhraní (API) poskytované knihovnou libpcap.

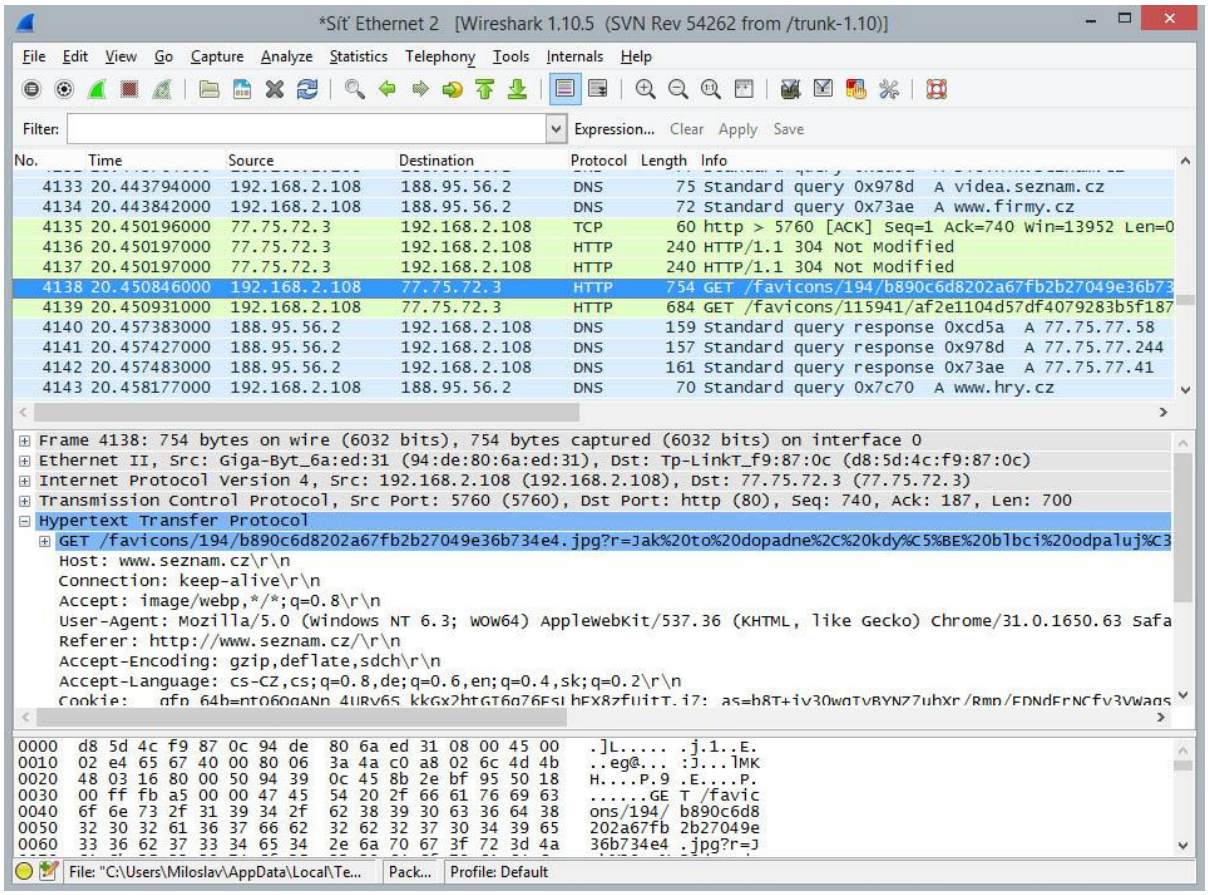
Datový formát souboru, do kterého obě knihovny umožňují ukládat zachycené pakety a později je načítat pro další zpracování reprezentuje formát PCAP (packet capture). Soubory s tímto formátem bývají označeny koncovkou .pcap, .cap, nebo .dmp [4] a mají MIME typ application/vnd.tcpdump.pcap [4].

[1] Délka dat je dána velikostí UDP paketu mínus velikost UDP hlavičky.
[2] http://www.tcpdump.org/
[3] http://www.winpcap.org/

2.2.1.1 Program Wireshark

Wireshark je program určený k zachytávání síťové komunikace a následné analýze protokolů pomocí mnoha implementovaných protokolových analyzátorů (mohou být vytvořeny dodatečné pluginy). Využívá knihoven libpcap (Linux, BSD, MacOS X) a WinPcap (MS Windows), které potřebuje ke svému provozu.

Program umožňuje zachytávat síťovou komunikaci na zvolených síťových rozhraních, provádět jejich následnou analýzu a ukládání do různých formátů. Ihned po zachycení jsou data programem zpracována a uživatel má tak k dispozici zdrojovou, cílovou adresu a použitý komunikační protokol, pokud je program spuštěn s GUI[1]. Zobrazovány jsou pouze záznamy, které odpovídají aktuálnímu použitému filtru, který je možné měnit i při spuštěném odchytávání. Díky implementaci mnoha protokolových analyzátorů je program schopen zobrazovat zapouzdření síťových protokolů. Další využití programu je analýza již zachycených dat, které jsou uloženy v souboru např. PCAP soubory. Umožňuje také přepnout síťové rozhraní do promiskuitního režimu, pokud jej rozhraní podporuje a zachytávat tak veškerou komunikaci na médiu.



Obrázek 2.4: Ukázka programu Wireshark

[1] Graphical user interface – grafické uživatelské rozhraní

2.2.1.2 Zachycená síťová komunikace ve formátu PCAP

Jedná se o jednoduchý formát pro ukládání zachycených paketů na síti, který se prakticky stal základem všech programů na zachytávání síťové komunikace [5]. To je způsobeno tím, že s ním přišla knihovna libpcap, která je ve velké míře využívána programy pro analýzu síťové komunikace.

Obrázek 2.5: Struktura formátu PCAP

Struktura tohoto formátu je vcelku jednoduchá, jak je vidět na obrázku Obrázek 2.5. Každý soubor v tomto formátu začíná polem *Global Header*, které na svém začátku nese čtyř bajtovou hodnotou. Tato hodnota se označuje jako magické číslo. Má hodnotu *0xd4c3b2a1* [5] a slouží pro identifikaci souboru v tomto formátu. Pole *Global Header* obsahuje poté ještě další hodnoty, které jsou pro zpracování tohoto formátu více či méně významné. Z nich bych vyzdvihnul hodnotu pro typ linkové vrstvy, kterou potřebujeme znát, abychom při zpracování dat dokázali správně rekonstruovat rámec na linkové vrstvě. Strukturu celého pole *Global header* zobrazuje tabulka *Tabulka 2*.5.

Bajty	0	1	2	3
0	Magické číslo (0xd4c3b2a1)			
4	Číslo verze (horní pozice)		Číslo verze (dolní pozice)	
8	Korekce lokálních hodin oproti času GMT[1].			
12	Přesnost časových razítek.			
16	Maximální velikost zachyceného paketu.			
20	Typ linkové vrstvy.			

Tabulka 2.5: Struktura pole Global Header

Za tímto první uvozujícím polem následují za sebou dvojice položek *Packet Header* a *Packet Data*. Každá dvojice je vždy pro jeden paket. Pole *Packet Header* obsahuje pouze časová razítka, velikost zachyceného paketu v poli *Packet Data* a původní velikost paketu. Velikost zachyceného paketu by neměla být větší než položka původní velikost paketu. Druhé pole *Packet Data* obsahuje vlastní zachycený paket. Struktura pole *Packet Header* je zobrazena v tabulce *Tabulka 2*.*6*

Bajty	0	1	2	3
0	Časové razítko zachycení paketu v sekundách.			
4	Časové razítko zachycení paketu v milisekundách.			
8	Počet oktetů v následujícím poli Packet Data.			
12	Velikost paketu			

Tabulka 2.6: Struktura pole Packet Header

[1] Greenwich Mean Time

2.2.2 Netflow

Netflow představuje technologii vyvinutou společností Cisco[1] jako doplňková služba určená pro export směrovacích tabulek. Později se jeho hlavní účelem stalo monitorování síťového provozu přes IP toky. Díky tomu poskytuje obraz provozu na síti v reálném čase. Proto ji také využívají poskytovatelé internetového připojení tzv. ISP (Internet Service Provider), jenž využívají nasbíraných statistik k účtování služeb. Dalšími uživateli jsou síťový administrátoři, kteří Netflow využívají pro sledování provozovaných aplikací na síti a další analýze síťového provozu. Mohou tak monitorovat chování aktivit uživatelů na síti. Výhodné je zejména pro odhalování bezpečnostních útoků (např. DoS). Také se dá použít k plánování sítě na základě statistik provozu.

2.2.2.1 IP tok

IP tok je v Netflow systému základní jednotka pro identifikaci síťového provozu. Jedná se o síťový tok, který byl popsán kapitole 2.1.6 a v systému Netflow je identifikován následující sedmicí hodnot:

- zdrojová a cílová IP adresa,
- zdrojový a cílový port,
- název logického rozhraní,
- protokol na vrstvě L3 IOS/OSI modelu,
- hodnoty ToS z IP hlavičky.

Pakety, které se tedy shodují v uvedené sedmici hodnot, jsou zahrnuty do jednoho toku, tak dlouho dokud nenastane exspirace. Ta nastane v jednom z následujících případů.

- Byl detekován konec toku (např. RST nebo FIN u protokolu TCP).
- Po stanovenou dobu nebyl tok aktivní (neaktivní timeout).
- Jedná se o příliš dlouhý tok (aktivní timeout).
- Došlo k zaplnění paměti Netflow cache.

2.2.2.2 Netflow architektura

Netflow architektura se skládá z několika prvků (hardwarových a softwarových), které získávají statistiky o tocích, sbírají tyto data a vizualizují je uživateli.

Exportér

Jedná se o síťové zařízení nebo software, který monitoruje procházející provoz a vytváří záznamy o tocích. V případě síťového zařízení se jedná o směrovač s podporou Netflow nebo o samostatné sondy. Pokud jde o Netflow exportér na směrovači, tak nejsou do informacích o tocích zahrnovány úplně všechny pakety, ale provádí se vzorkování. V případě, že by směrovač zaznamenával do toku každý paket, tak by to bylo nesmírně náročné na hardwarové nároky, aby se nesnížila rychlost směrování. Například deterministickým vzorkováním 1 z 5 bychom zaznamenávali 20% paketů, což redukuje požadavky na hardware. Pokud se jedná Netflow sondu, tak to je samostatné zařízení zapojené do sítě, které je transparentní pro provoz a zaznamenává informace o tocích.

Informace o tocích se na exportéru zaznamenávají do paměti Netflow cache. Buďto se vytvoří nový záznam nebo se aktualizuje starší. Tento záznam je v Netflow cache dokud neexspiruje. Exspirace může nastat po vypršení časovače (neaktivní to), časovač byl aktivní po určitou dobu (příliš dlouhý

[1] http://www.cisco.com

tok), bylo ukončeno spojení (pakety RST a FIN) nebo je plná Netflow cache. Poté se toky exportují do Netflow kolektoru agregované (Netflow verze 9) nebo neagregované (Netflow verze 5).

Kolektor

Reprezentován síťovým zařízení, které přijímá Netflow pakety, jenž zasílá jeden nebo více exportérů. Zatímco exportérů bývá zpravidla více, tak kolektor bývá pouze jeden. Jak již název napovídá, tak jeho úkolem je zpracovat přijaté záznamy o tocích a ukládat tyto data na disk, do databáze nebo jiného úložiště. Nad takto zpracovanými daty se poté může pomocí dostupných nástrojů dotazovat a získávat přehledné informace v tabulkách či grafech.

2.2.2.3 Netflow v9 záznam

Jak již bylo uvedeno, tak Netflow exportér vytváří záznamy v Netflow cache. Záznamy, které exspirovaly, tak zasílá Netflow protokolem kolektoru. Tento protokol prošel vývojem a jeho podoba se v jednotlivých verzích měnila. Jeho poslední evoluce dala vzniku verzi 9. Mezi hlavní přínosy této verze patří používání šablon. Díky tomu se formát stal snadno rozšiřitelný do budoucna. Šablony totiž popisují strukturu záznamů, a proto je možná dodatečná variabilita zavedením dalších šablon.

Základní struktura formátu Netflow v9 je zobrazena na obrázku Obrázek *2.6*. Skládá se s hlavičky paketu zvané *Packet Header*, která je následována jedním nebo více bloky *Template FlowSet* a *Data FlowSet*. Hlavička *Packet Header* je převzatá z Netflow verze 5 [6]. Obsahuje číslo verze protokolu, sekvenční číslo, čas začátku a konce IP toku, číslo zařízení odesílají paket a počet následujích polí *FlowSet*.

Obrázek 2.6: Struktura záznamu Netflow v9

Pole *FlowSet* pro *Template* i *Data* je založeno na stejné struktuře. K identifikace, jestli se jedná o *Template FlowSet* nebo *Data FlowSet* slouží 16 bitová hodnota na začátku pole *FlowSet* nazývaná *FlowSet ID*. *Template FlowSet* popisuje počet položek, jejich typ a délku. Obsahuje tedy pouze metadata, která popisují strukturu dat v následujících polí *Data FlowSet*.

3 Prostředky distribuovaného zpracování velkých dat

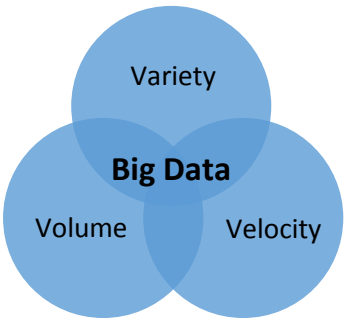
3.1 Distribuované výpočty a Big Data

Aplikace často vyžadují pro svůj běh více zdrojů, jako je více paměti RAM či rychlejší procesor, než je dostupný na počítači, na kterém běží. Tento problém nastává v mnoha organizacích, které jej mohou vyřešit zakoupením silnějšího stroje s větší pamětí a rychlejším procesorem nebo pořízením vlastního výpočetního clusteru, který se bude na venek tvářit jako jeden počítač. Výpočet bude na clusteru probíhat paralelně, takže se rozdělí na jednotlivé uzly clusteru. Jednotlivé uzly clusteru nemusí být tak výkonné, protože se výpočet rozdělí a můžou se skládat z běžně dostupného hardware, jehož cena je mnohem nižší než špičkové komponenty do jedno příp. dvou procesorové sestavy. Pořízení vlastního clusteru je ovšem cenově nákladná věc, a tak jako alternativa se nabízí využití cloudových služeb. V těchto službách si uživatel může vytvořit a nakonfigurovat výpočetní stoje příp. clustery dle své potřeby. Platí přitom většinou za využitý procesorový čas nebo za přenesená data v případě databází. Má ovšem jistotu, že jeho data jsou zálohována napříč servery v datovém centrum a nemusí řešit výpadky či poruchy hardware. Takováto cloudová centra dnes provozuje mnoho poskytovatelů. Mezi nejznámější patří Amazon Web Services[1], Windows Azure[2] či Google Cloud Platform[3].

Pro zpracování velkých dat tzv. Big Data prakticky neexistuje jiný přístup než použít distribuovaný výpočet, pomocí nějž zpracujeme tyto data v rozumném čase. Pracuje se zde totiž se soubory dat, jejichž velikost je mimo schopnosti běžných softwarových nástrojů tyto data zachycovat, spravovat a zpracovávat [7]. Jejich velikost se pohybuje v řádech giga, tera či peta bajtů, které jsou díky své velikosti obvykle ukládány v datových skladech a jsou volně strukturované či nestrukturované.

Při zpracování velkých dat se nelze omezit pouze na zvětšení prostoru pro jejich uložení, ale je nutné brát v úvahu i jejich další charakteristiky, které se označují 3V [7]. Pojem 3V vychází z počátečních písmen anglických slov volume (objem), velocity (rychlost) a variety (různorodost), jenž Big Data charakterizují.

- Volume – Objem dat narůstá exponenciálně.
- Velocity – Zvyšuje se rychlost, s jakou data vznikají včetně potřeby jejich analýzy.
- Variety – Vzrůstá různorodost typů dat (strukturovaná až nestrukturovaná data).



Obrázek 3.1: Charakteristiky Big Data

[1] http://aws.amazon.com/
[2] http://www.windowsazure.com/
[3] https://cloud.google.com/

3.2 Výpočetní model MapReduce

MapReduce je výpočetní model, který byl představen společností Google [8], jak metoda na zpracování dat na clusteru, který se skládá z počítačů sestavených z běžně dostupného hardware. Je založený na rozdělení zpracovávaných dat na velké množství výpočetních uzlů čímž dosáhneme velkého výpočetního výkonu a rychlého zpracování. Toto rozdělení nemusí být vždy úplně jednoduché, ale pokud jej dosáhneme, tak získáme dobře škálovatelnou aplikaci, jejíž výpočet může běžet na stovkách, tisících nebo až desetitisících výpočetních uzlech. Díky této jednoduché škálovatelnosti vzbudil pozornost mezi mnohými programátory [9].

Programy využívající výpočetní model MapReduce jsou vykonávány ve dvou hlavních fázích, nazývaných *mapping* a *reducing* [9], jejichž činnosti jsou definovány ve funkcích *Map* a *Reduce*. Během provádění programu se nejprve paralelně na výpočetních uzlech provádí fáze mapping a až po jejím skončení nastupuje fáze reducing. Fáze mapping přijme část vstupních dat, ze kterých si určí jejich klíč příp. je dále zpracuje a zasílá do další fáze reducing klíč a data. Ne všechny vstupy do fáze mapping musí produkovat výstupy do fáze reducing. Nejčastěji se ve fázi mapping provádí filtrování a transformace dat pro fázi reducing. Do fáze tedy projdou data, která jsou pro daný problém užitečná. Fáze reducing začíná po skončení fáze mapping, kdy jsou již vstupní data funkcí Map roztříděna. Vstupem do funkce Reduce je dvojice klíč a seznam hodnot. Tyto klíče byly vytvořeny ve fázi mapping a seznam hodnot obsahuje všechny hodnoty z fáze mapping, které byly v této fázi označeny daným klíčem. Ve fázi reducing se poté provádí obvykle agregace výsledků pro hodnoty se stejným klíčem a odeslání výstupních hodnot do výsledku.

Kromě fází mapping a reducing jsou prováděny při MapReduce programu ještě fáze *partitioning* a *shuffling*. Fáze partitioning rozděluje vstupní data a zasílá je na jednotlivé výpočetní uzly do funkce Map. Fáze *shuffling* agreguje výstupy se stejným klíčem z fáze mapping a předává je do fáze reducing.

Funkce	**Vstup**	**Výstup**
Map	k1:v1	list(k2:v2)
Reduce	k2:list(v2)	list(k3:v3)

Tabulka 3.1: Srovnání vstupních a výstupních hodnot MapReduce výpočetního modelu.

3.2.1 Ukázka MapReduce aplikace

Mějme textový vstupní soubor, pro který chceme vypočítat četnost výskytu jednotlivých slov. Začneme vytvořením funkce `Map`, jejíž vstupy budou číslo řádku načteného ze souboru jako klíč a hodnota na daném řádku v souboru. Ve funkci `Map` se provede rozdělení řádku ze souboru na jednotlivá slova. Každé slovo se poté emituje (zasílá) do fáze reducing, kde klíčem je dané slovo a hodnotou je celočíselná hodnota *1*, která značí jeden výskyt daného slova. Funkce `Map` zapsaná v pseudokódu je následující.

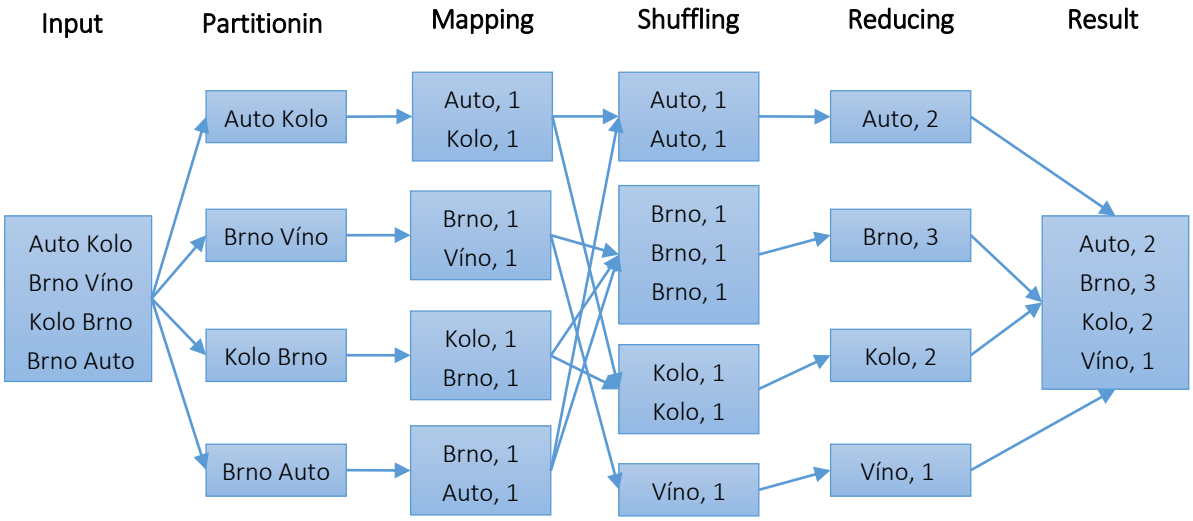
```
Map(Integer key, String line) {
   List<String> words = split(line, ' ');
   for each word in words {
      emit ((String) word, (Integer) 1);
   }
}
```

Jakou druhou vytvoříme funkci `Reduce`, která bude ve svých parametrech přijímat klíč a seznam hodnot. Jako klíč bude vstupovat jedno slovo ze souboru a v seznamu hodnot budou jeho jednotlivé výskyty. Úkolem funkce `Reduce` bude tyto výskyty agregovat a vypočítat tak celkový počet výskytů daného slova. Jedna funkce `Reduce` bude vždy počítat počet výskytů pro jedno slovo. Pseudokód funkce `Reduce` bude následující.

```
Reduce(String token, List<Integer> values) {
   Integer sum = 0;
   for each value in values {
      sum = sum + value;
    }
    emit ((String)token, (Integer) sum);
}
```

Na uvedeném příkladu výpočtu četnosti jednotlivých slov v souboru je demonstrována „elegance“ přístupu MapReduce. Pomocí dvou krátkých funkcí dokážeme tímto přístupem vyřešit daný problém jednoduše, za použití distribuce výpočtu. V příkladu se dále předpokládá využití frameworku (např. Apache Hadoop), který podporuje model MapReduce a fáze partitioning a shuffling řeší implicitně za programátora.

Činnost MapReduce programu počítajícího četnost slov je ilustrována na obrázku *Obrázek 3.2*.



Obrázek 3.2: Ilustrace MapReduce programu počítajícího četnost slov.

3.3 Apache Hadoop

Apache Hadoop[1] je open source framework vyvíjený společností Apache Software Foundation. Obsahuje sadu softwarových komponent napsaných v jazyce Java, které jsou určeny pro zpracování nestrukturovaných velkých dat. Využívá k tomu MapReduce výpočetní model popsaný v kapitole 3.2. Jeho podstatou je poskytnout softwarové nástroje pro vytváření aplikací, které budou dobře škálovatelné za použití distribuce. Škálovatelnost je navržena, tak že aplikace nad ním vytvořené mohou běžet na jednom serveru až po tisíce serverů, kdy každý server bude provádět část výpočtu a ukládat část dat. Přičemž jednotlivé servery nemusí být postaveny ze speciálního hardware, ale z běžně dostupného. Pro ukládání dat používá Apache Hadoop vlastní distribuovaný souborový systém HDFS (Hadoop Distributed File System).

[1] http://hadoop.apache.org/

3.3.1 Souborový systém HDFS

MapReduce je pouze model výpočtu, který neřeší, kde budou data uložena. Pro distribuované uložení dat nabízí Apache Hadoop virtuální souborový systém HDFS, který vznikl z GFS (Google File System) od společnosti Google. Tento souborový systém distribuuje data a jednotlivé uzly zapojené do Apache Hadoop clusteru. Metadata popisující tato data jsou přitom uložena mimo vlastní data na jednom uzlu. Řeší přitom optimální uložení dat, výkonnost a odolnost vůči výpadkům. Jelikož se ovšem jedná od souborový systém virtuální, tak je postaven nad běžnými souborovými systémy jednotlivých uzlů. Proto neřeší fyzické uložení dat na uzlu, ale obstarává nalezení úložiště a přístup k datům. Přístup je sekvenční (čtení a zápis), protože MapReduce je dávkové zpracování často velkých dat (BigData), pro které byl navržen. Což značí, že samotné čtení/zápis jsou rychlé operace a většina režie spadá na nalezení dat.

Uložení dat je řešeno pomocí bloků fixní velikosti (typicky 64 nebo 128 MB), které jsou distribuovány na jednotlivé uzly clusteru. Jednotlivé bloky nejsou uloženy pouze v jedné kopii, ale jsou replikovány na další uzly pro zajištění redundance. Fyzicky je jeden bloku uložen jako několik bloků souborového systému na daném uzlu, a proto pokud je soubor nezabere celý blok mu vyhrazený v HDFS, tak zabere v lokálním souborovém systému pouze svoji „téměř“ skutečnou velikost. Naproti tomu u lokálních souborových systémů, pokud soubor nezaplní celý blok, tak na disku celý blok obsadí, protože je mu přiřazen.

3.3.1.1 NameNode

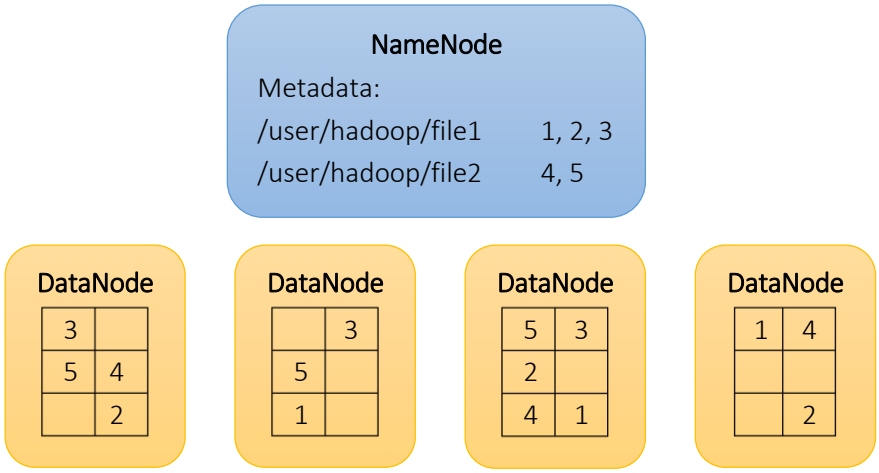
Souborový systém HDFS je postaven na architektuře master-slave. NameNode je master uzel souborového systému HDFS, který spravuje souborový systém a metadata souborů (adresáře, cesty k souborům, jejich atributy a místa uložení). Tento uzel je v clusteru jediný a bývá umístěn na výkonném a spolehlivém uzlu, který neobsahuje uživatelská data ani neprování výpočet MapReduce programů. Bývá totiž intenzivně vytížen vstupními/výstupními operacemi, protože poskytujeme metadata o souborech pro celý cluster. Pokud chce klient přečíst soubor uložený v HDFS, tak kontaktuje NameNode uzel, který obsahuje meta informace ve kterých blocích a na kterých uzlech je daný soubor uložen. Jeho nevýhoda je v tom, že je pouze v jedné instanci a při jeho výpadku je ovlivněn celý Apache Hadoop cluster, jelikož nemůže být automaticky zastoupen jiným uzlem, jako je tomu v případě datových uzlů, když jsou soubory uloženy ve více kopiích.

3.3.1.2 DataNode

NameNode reprezentuje master uzel souborové systému HDFS a ostatní slave uzly se nazývají DataNode. DataNode uzel obsahuje vlastní bloky dat souborů v HDFS systému. Když chceme zapsat nebo přečíst soubor z HDFS, tak je tento soubor rozdělen do bloků a NameNode uzel klientovi sdělí kam daný blok umístit. Klient poté komunikuje přímo z DataNode uzly. Po uložení souboru pro zvýšení redundance mezi sebou komunikují jednotlivé DataNode uzly a replikují své soubory mezi sebou.

Uložení souborů v HDFS využívající uzlu NameNode a uzlů DataNodes je ilustrováno na obrázku *Obrázek 3.3*. Na tomto obrázku je ukázáno uložení dvou souborů, z nichž první je */user/hadoop/file1* a druhý */user/hadoop/file2*. Soubor *file1* je rozdělen do tří bloků označených čísly 1, 2 a 3, a soubor *file2* do bloků označených 4 a 5. Tyto jednotlivé bloky jsou rozdistribuovány přes uzly DataNode z nichž každý je umístěn ve třech kopiích pro zajištění redundance. Pokud tedy některý z uzlů bude mimo provoz nebo bude chvíli nedostupný, tak bude zastoupen některým jiným uzlem a všechny soubory budou dostupné.

Aby master uzel NameNode věděl, které bloky jsou kde uloženy, potřebuje informace od DataNode uzlů. Tyto uzly jej proto od inicializace informují o blocích, které ukládají. Jakmile proběhne tato výměna metadat, tak se DataNode uzel začne NameNode uzlu dotazovat na lokální změny stejně tak jako na nové soubory, jejich přesunutí či smazání, ke kterým došlo na jiných zlech.



Obrázek 3.3: Příklad distribuce souborů v HDFS.

3.3.1.3 Secondary NameNode

Secondary NameNode, dále jako SNN, je démon, který monitoruje stav distribuovaného souborového systému HDFS v clusteru. Stejně jako uzel NameNode, tak je i SSN je pouze jediný na cluster a obvykle běží na vlastním stroji [9]. Slouží pro ukládání stavu NameNode uzlu, kdy si jej v definovaných intervalech ukládá. Uzel tedy neobsahuje pokaždé stejná data jako uzel NameNode. Snaží se ale díky uloženému stavu minimalizovat dobu přístupu k datům a ztrátu dat, v případě výpadku NameNode uzlu. Při výpadku NameNode uzlu je nutné provést ruční rekonfiguraci clusteru, aby se SSN začal používat jako NameNode master uzel.

3.3.2 Architektura Hadoop

Stejně jako architektura distribuovaného souborového systému HDFS, tak i výpočetní část Apache Hadoop clusteru je založena na master-slave architektuře. V předchozí kapitole jsme zmínily uzly, které obstarávají distribuovaný souborový systém HDFS. Pro provádění výpočtu MapReduce úloh jsou určeny démoni

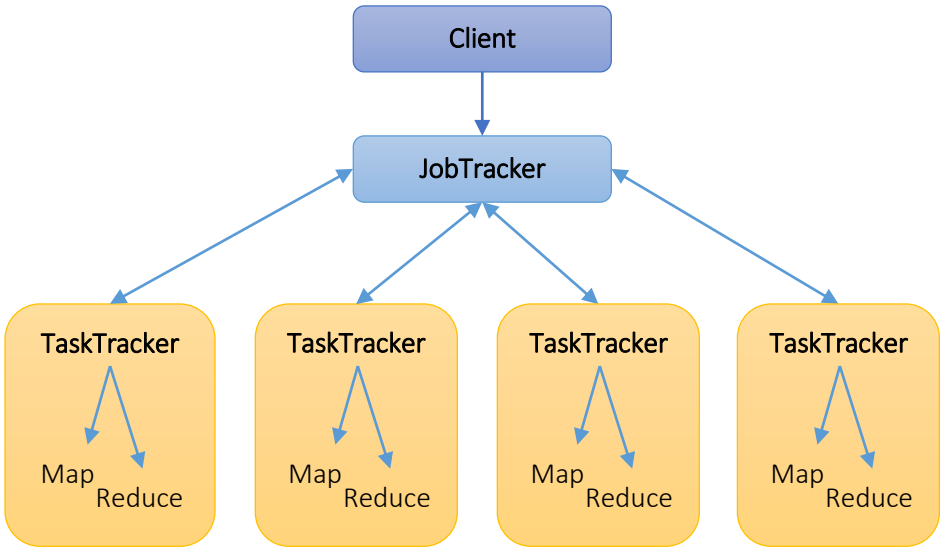
- JobTracker,
- TaskTracker.

3.3.2.1 JobTracker

Démon JobTracker reprezentuje master výpočetní uzel v Hadoop architektuře. Představuje prostředníka mezi MapReduce aplikací a Hadoop clusterem. Přijímá všechny požadavky na výpočet MapReduce úloh, pro které určí plán provádění. Tento plán obsahuje soubory, které budou úlohou zpracovávány a přidělené výpočetní uzly. Během výpočtu úlohy ji monitoruje a zjišťuje její aktuální stav. Pokud nastane případ, že úloha není dokončena, tak ji může znovu spustit na jiných výpočetních uzlech. Maximální nastavený počet spuštění úlohy je přitom dodržen.

3.3.2.2 TaskTracker

V HDFS systému se jednotlivá data ukládají na slave uzly. Stejně je tomu i pro výpočetní část, kdy výpočet provádí slave uzly zvané TaskTracker. Přitom jsou řízeny master uzlem JobTracker. Ten dohlíží na celkové provádění úlohy a jednotlivým slave uzlům přiděluje individuální úlohy, které budou počítat. Přitom každý TaskTracker je zodpovědný za individuální úlohy, které mu byly JobTrackerem přiděleny. Může mu být současně předěleno více Map příp. Reduce částí spadajících do fáze mapping příp. reducing. Komunikace mezi klientem, JobTrackerem a TaskTrackery je znázorněna na obrázku *Obrázek 3.4*.



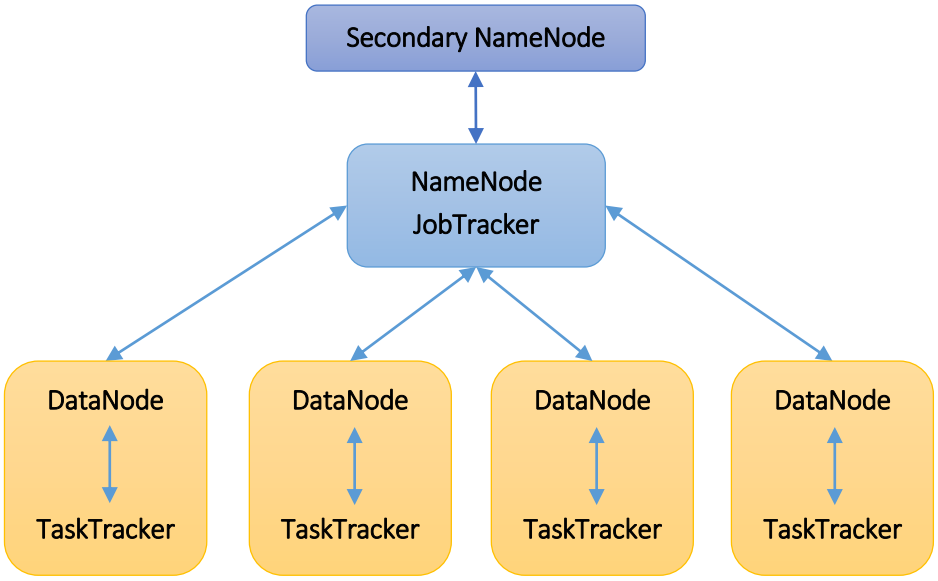
Obrázek 3.4: Schéma komunikace JobTrackeru a slave TaskTrackery.

Zodpovědností TaskTrackeru je neustále komunikovat s JobTrackerem. Pokud se tato komunikace na určitou dobu přeruší, tak to pro JobTracker značí, že daný TaskTracker není dostupný. V tom případě JobTracker úlohy, které zpracovával daný TaskTracker, odešle ke zpracování na jiné uzly v clusteru.

3.3.2.3 Hadoop cluster

Konfigurace Hadoop clusteru se skládá z démonů zajišťujících činnost distribuovaného souborového systému HDFS a démonů výpočetní části. Na obrázku *Obrázek 3.5* je ukázána typická konfigurace Hadoop clusteru.

Tento cluster je rozdělen na master (modrá barva) a slave uzly (žlutá barva). Na master uzlu jsou spuštěni démoni NameNode a JobTracker. Na samostatném uzlu dále běží démon pro Secondary NameNode, který nemůže být spuštěn na stejném uzlu jako NameNode démon, protože provádí jeho zálohu pro případ výpadku. V případech že máme cluster malý, tak může Secondary NameNode běžet i na některém ze slave uzlů. Na slave uzlech jsou spuštěny démoni DataNode a TaskTracker. Vždy na jednom slave uzlu běží společně jeden DataNode a TaskTracker. To je výhodné, protože TaskTracker komunikuje přímo z DataNode démonem na stejném uzlu a nemusí vždy komunikovat přes propojovací síť clusteru s jinými uzly DataNode.



Obrázek 3.5: Konfigurace Hadoop clusteru

3.3.3 Hadoop Streaming

Jak je zmíněno v kapitole 3.3, tak Apache Hadoop je napsán v jazyce Java. Proto jsou vytvořeny Java balíčky podporující vytváření MapReduce aplikací pod tímto framworkem. Apache Hadoop totiž používá vlastní datové typy (třídy), umístěné v balíku org.apache.hadoop.io. Chceme-li vytvořit MapReduce aplikaci, tak vytvoříme mapper děděním z třídy org.apache.hadoop.mapreduce.Mapper a reducer děděním od třídy org.apache.hadoop.mapreduce.Reducer. V Mapperu implementujeme metodu Map a v Reduceru metodu Reduce. Jako parametry těchto metod se, ale nepoužívají standardní Java typy, ale právě datové typy z balíku `org.apache.hadoop.io`.

Chceme-li zpracovávat MapReduce úlohy, kde mapper a reducer budou psány v jiném programovacím jazyce než jazyce Java, tak hovoříme o tzv. streamovaných úlohách. Tyto úlohy se spouští přes Java program Hadoop Streaming (hadoop-streaming.jar), který je součástí instalace Apache Hadoop. Spouštíme jej v konzoli a přes jeho parametry konfigurujeme MapReduce úlohu, která se má provést. Přičemž parametry rozdělujeme na obecné (genericOptions) a steamovací (streamingOptions). Při spuštění musíme nejdříve zadávat obecné parametry následované parametry streamovacími. Toto pořadí je závazné, takže v případě jeho nedodržení se nejedná o platné spuštění. Rozdělení na dva druhy parametrů je z důvodů generalizace. Obecné parametry se totiž používat napříč Hadoop frameworkem různými aplikacemi a utilitami. Druhé streamovací parametry jsou specifické pouze pro Hadoop Streaming. Pro platné spuštění MapReduce úlohy musíme zadat minimálně povinné parametry, jejichž význam je specifikován v tabulce *Tabulka 3.2*.

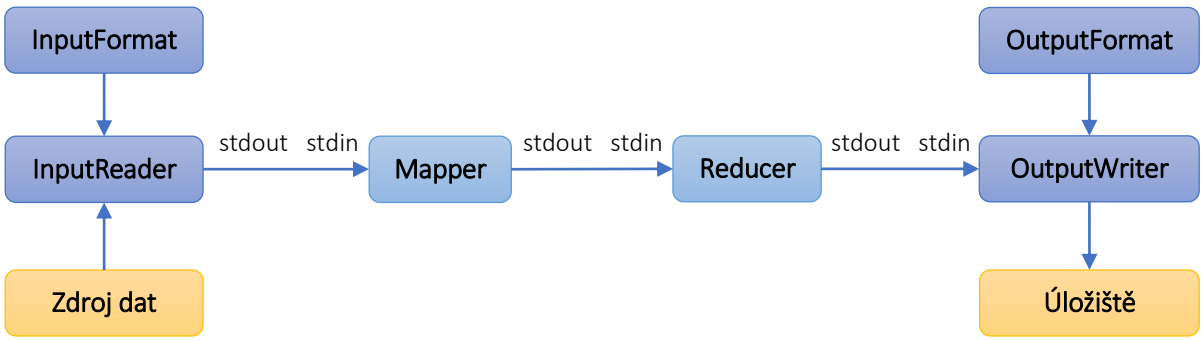
Parametr	**Hodnota parametru**	**Popis**
-input	vstupní adresář nebo soubor	zdroj dat pro mapper
-output	výstupní adresář	výstupní úložiště pro reducer.
-mapper	spustitelný program nebo JavaClassName	provede fázi mapping
-reducer	spustitelný program nebo JavaClassName	provede fázi reducing

Tabulka 3.2: Povinné parametry pro Hadoop Streaming.

Datová komunikace během zpracování MapReduce úloh přes Hadoop Streaming je založena na proudech standardního vstupu (STDIN) a standardního výstupu (STDOUT). Díky využití tohoto komunikačního kanálu může fázi mapping a reducing provádět program napsaný v libovolném programovacím jazyce. Jedinou podmínkou je, že program musí být spustitelný na operačním systému, na němž je Apache Hadoop nainstalován.

Diagram zpracování streamované úlohy využívající Hadoop Streaming je zobrazen na obrázku Obrázek *3.6*. Na počátku zpracování máme zdroj dat, který je specifikován parametrem `-input`. Tento zdroj dat čte *InputReader*, který je vytvořen blokem *InputFormat*. V obou případě se jedná o Java třídy. Díky tomuto přístupu můžeme zpracovávat různá vstupní data použitím vhodného vstupního formátu. Ve výchozím nastavení pokud nespecifikujeme jiný vstupní formát, tak se použije třída `org.apache.hadoop.TextInputFormat`, která zpracovává vstup jako text po řádcích. Na standardní výstup tedy posílá jednotlivé řádky textu, které naopak ze standardního vstupu čte *Mapper* a dále zpracovává. Stejným způsobem funguje i *Reducer*, který na výstup zasílá výsledná zpracovaná data, která následně do datového úložiště zapisuje *OutputWriter* vytvořen v bloku *OutputFormat*. Stejně jako v případě vstupu, tak i na výstupu jsou OutputFormat a OutputWriter Java třídy. Ve výchozím nastavení se jedná stejně jako na vstupu o textové zpracování, které zajišťuje třída `org.apache.hadoop.TextOutputFormat`. Vlastní formáty pro vstup a výstup se zadávají jako JavaClassName do streamovacích parametrů `-inputformat` pro vstup resp. `-outputformat` pro výstup.

Z definice povinných parametrů pro Hadoop Steaming uvedených v tabulce Tabulka *3.2* se může zdá, že vstupem a výstupem může být pouze soubor resp. soubory. Není tomu ovšem tak, protože při implementaci vlastního vstupního a výstupního formátu nemusíme hodnoty z povinných parametrů pro určení vstupu a výstupu využívat. Můžeme místo toho využít definic v obecných parametrech a předat, tak například připojení k databázi, kterou můžeme využívat pro vstup a výstup.



Obrázek 3.6: Diagram zpracování přes Hadoop Streaming.

3.3.4 Apache Hadoop distribuce

Společnost Apache Software Foundation[1] vyvíjí projekt Apache Hadoop a nástroje v něm obsažené. Vydává jej pod licencí Apache License, která umožňuje takto vytvořený software používat svobodně za jakýmkoliv účelem bez nároků na licenční poplatky. To umožňuje dalším společnostem tuto, můžeme říci referenční implementaci, dále upravovat přidáváním vlastních nástrojů a dále distribuovat. Dostupných je tedy mnoho distribucí mezi, kterými najdeme i komerční placené nástroje. Pro naši potřebu ovšem připadají vhodné nástroje pro platformu Windows, které si představíme.

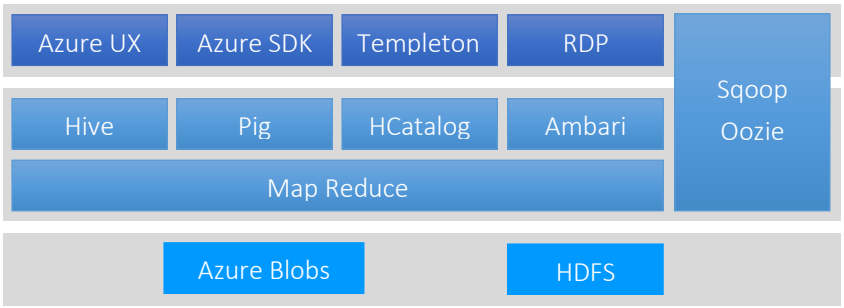
[1] http://www.apache.org/

3.3.4.1 Hortonworks Data Platform for Windows

Jedná se o zástupce open-source distribuce Apache Hadoop, která je dostupná i pro platformu Microsoft Windows Server. Tato distribuce byla poprvé k dispozici jako beta verze a později se dočkala vydání stabilní verze. Vznikla na základě partnerství se společností Microsoft uzavřeného v říjnu 2011 [10]. Hortonworks spolupracuje se společností Microsoftem na vývoji Apache Hadoop distribuce do cloudu Windows Azure a na operační systém Windows server.

3.3.4.2 Windows Azure HDInsight

HDInsight představuje Apache Hadoop běžící jako služba v cloudu Windows Azure společnosti Microsoft. Je založena Hortonworks Data Platform[1] [11], což je open source distribuce Apache Hadoop, které je dostupná i pro platformu Windows. Struktura služby HDInsight je zobrazena na obrázku *Obrázek 3.7*.



Obrázek 3.7: Struktura Windows Azure HDInsight

[1] http://hortonworks.com/

4 Návrh řešení pro distribuované zpracování velkých dat

Kapitola se zabývá návrhem aplikace, která bude provádět zpracování síťových dat a získávání informací z nich. Využito přitom bude distribuovaného zpracování vstupních síťových dat, které bude založeno na výpočetním MapReduce. Dále návrh počítá s rozdělením aplikace do dvou funkčních částí. V první části se budou zařazovat vstupní soubory se síťovými daty ke zpracování na clusteru Hadoop. Druhá část potom bude využívat informací ze zpracovaných síťových dat z první části a bude se moci nad nimi dotazovat. Výsledky dotazů budou zobrazovány v textové nebo grafické podobě a budou uživateli poskytovat informace o provozu na dané síti.

4.1 Zpracování PCAP souborů přes MapReduce

Vstupní data se zachycenou síťovou komunikací uloženou ve formátu PCAP se začínají zpracovávat na klientovi, který komunikuje s Apache Hadoop clusterem. Klient nejprve nahraje zvolený PCAP soubor do Hadoop clusteru, což představuje jeho uložení do distribuovaného úložiště HDFS. Následně vytvoří novou MapReduce úlohu se vstupním souborem nahraným do HDFS úložiště a zařadí ji do fronty úloh k vykonání.

Vlastní MapReduce úloha na zpracování PCAP souboru bude pracovat na následujícím principu. Ze vstupního PCAP souboru uloženého v HDFS se budou číst jednotlivé pakety, které se následně budou zasílat do fáze mapping zpracování MapReduce úlohy. Zde se identifikuje klíč konverzace, který se bude zasílat s daty z aplikační vrstvy do další fáze. Vstupem do fáze reducing bude tedy klíč a data z aplikační vrstvy ze všech paketů označených daným klíčem. Jejím úkolem potom bude identifikovat aplikační protokol, zpracovat jej a uložit výsledek na výstup MapReduce úlohy. Klíč extrahovaný ve fázi mapping bude identifikovat konverzace na základě pětice položek:

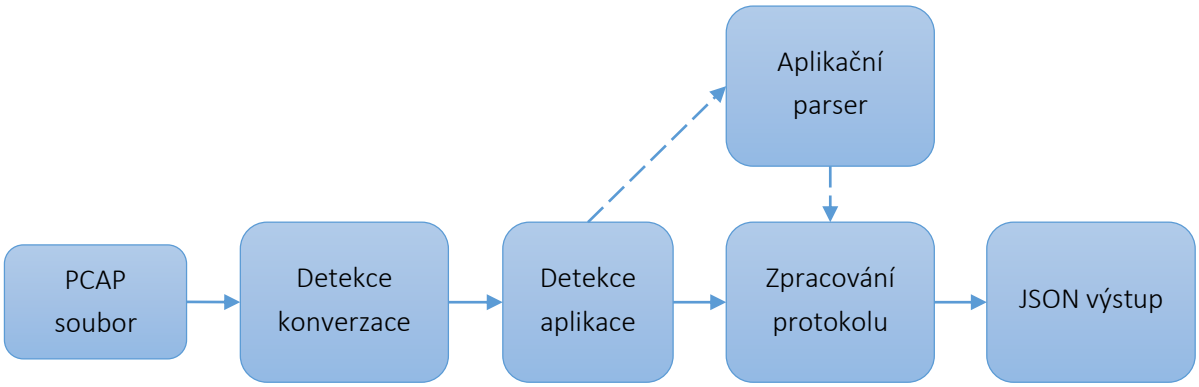
- zdrojová IP adresa,
- cílová IP adresa,
- zdrojový port,
- cílový port,
- protokol na transportní vrstvě.

Pro získání toho klíče se provede rozbalení přes jednotlivé datové jednotky, tak jak jsou v sobě zapouzdřeny dle kapitoly 2.1.2. Nejdříve se z paketu vytvoří datová jednotka na linkové vrstvě dle typu linkové vrstvy. Z ní se následně získá IP datagram, v němž je uložena zdrojová a cílová IP adresa. V IP datagramu je v datové části uložen protokol na vyšší vrstvě, což bude většinou TCP nebo UDP paket případně jiný paket s transportním protokolem. Tento protokol na transportní vrstvě identifikujeme a získáme z něj zdrojový a cílový port. Tímto zpracováním dostaneme všechny hodnoty pro stanovení klíče konverzace. Může ovšem nastat, že se nejedná o komunikaci na transportní vrstvě, jde například o komunikace protokolem ICMP. V takové případě bude protokolem v klíči uveden protokol ICMP a místo portů bude identifikátor ICMP relace.

Druhá fáze reducing bude v jedné funkci Reduce zpracovávat jednu konverzaci dle daného klíče. Z klíče se identifikuje aplikace pomocí dobře známých a registrovaných portů. Po identifikaci aplikace se vybere příslušný aplikační parser na zpracování aplikačního protokolu, který aplikace

používá na aplikační vrstvě. Výstup MapReduce se bude ukládat do výstupního souboru ve formátu JSON (JavaScript Object Notation), který bude uložen v HDFS a poté importován do NoSQL databáze.

Během konfigurace MapReduce úlohy bude aplikace nabízet volbu rozsahu zpracování vstupního souboru. Bude možné zvolit, jestli se mají aplikace pouze detekovat nebo detekovat a zpracovávat jejich aplikační dat. Jednotlivé aplikační parsery se budou do aplikace přidávat pomocí pluginů. Díky tomu bude zaručená rozšiřitelnost.



Obrázek 4.1: Architektura zpracování PCAP souboru.

4.2 Dotazování se nad zpracovanými daty

Druhá část aplikace bude poskytovat klienta, který bude sloužit pro vizualizaci a analýzu dat ze zpracovaných PCAP souborů na Hadoop clusteru. Vizualizace bude provedena pomocí grafů, které budou zobrazovat získané informace ze síťových toku. Jako příklad grafů mohu uvést statistiku používaných aplikačních protokolů či nejčastější zdrojové a cílové IP adresy. Analýza dat bude založena tabulce, která bude zobrazovat jednotlivé síťové toky s uvedením počtu zpráv v daném síťovém toku a čas začátku a konce datového toku. Tato tabulka bude umožňovat řazení a filtrování dle jednotlivých sloupců. Každý identifikovaný síťový tok tedy bude reprezentován jedním záznamem v tabulce a bude pro něj možné zobrazit jeho detail. Detail síťového toku bude poté zobrazovat v tabulce jednotlivé zprávy a získaném informace z nich, které byly získány z výstupu aplikačního parseru.

5 Implementace řešení

Po prozkoumání problematiky datové komunikace, možností jeho jejího zachycování a následného zpracování bylo z těchto poznatků navrženo řešení pro zpracování síťových dat. Toto řešení, bylo odzkoušeno na jednoúčelové konzolové aplikaci. Bylo tím ověřeno možné využití frameworku Apache Hadoop, ale nemuselo řešit řadu problémů, na které se později narazilo. Ověření totiž spočívalo pouze ve spuštění jednoduché MapReduce úlohy, která pouze identifikovala síťové a dále již neprováděla další zpracování. Na problémy a důležité části z implementace je zaměřena právě tato kapitola.

Výsledná aplikace byla implementována jako aplikace s grafickým uživatelským rozhraním. Tato aplikace umožňuje zadávat PCAP soubory se zachycenou síťovou komunikací ke zpracování přes výpočetní model MapReduce na Apache Hadoop clusteru. Dalším funkčním celkem je poskytování výsledků tohoto zpracování s možnostmi pro následnou analýzu a získávání dalších informací o zachycené komunikaci.

Implementace výsledné aplikace využívá několik technologií. Jejich výběrem a popisem začíná tato kapitola. Dále pokračuje v popis již vlastního implementovaného řešení. Je zde uvedeno, jak byla aplikace oddělena do logicky souvisejících celků. Jaký způsobem bylo vyřešeno grafické uživatelské rozhraní a jaký návrhový model využívá. Po této části se již kapitola zaměřuje implementaci zpracování PCAP souboru se zachycenou síťovou komunikací na Apache Hadoop clusteru s využití programovací modelu MapReduce. V této části je uvedena implementovaná struktura tohoto zpracování s rozebráním jednotlivých fázi. V těch jsou uvedeny jednotlivé koncepty a způsob zvoleného řešení. Kapitola je poté ukončena použitou strukturou v databázi MongoDB, do které se přímo ukládají výsledky MapReduce úloh.

5.1 Použité technologie

Při výběru technologií, které budou použity pro vývoj výsledné aplikace, jsem vycházel ze zadání, které specifikovalo jako implementační jazyk libovolný .NET jazyk. Po dohodě s vedoucím práce jsem se na základě podmínky, aby se jednalo o objektově orientovaný jazyk a vlastních zkušeností rozhodl, že výsledná aplikace bude napsána v jazyce C# (čteno anglicky jako „C sharp“) a bude koncipována jako Winows Presentation Foundation aplikace. Během vývoje jsem však musel použít i jazyk Java, abych dosáhnul potřebné funkčnosti. Jakmile byl zvolen programovací jazyk, tak bylo nutné ověřit, jestli pro něj existuje nějaká podpora pro práci s Apache Hadoop clusterem. Bylo nalezeno SDK (software developement toolkit) Microsoft .NET SDK For Hadoop[1] distribuované pod volnou licencí Apache License 2.0. Toto SDK se zpočátku zdálo jako dostačující, ale postupem času se stále více naráželo na jeho limity.

Pro ukládání zpracovaných dat připadaly v úvahu různé NoSQL databázové systémy. Již na počátku zpracování bylo jasné, že nepůjde použít (nebo by to bylo velice komplikované) relační databáze, protože bude potřeba ukládat různé hodnoty pro různé aplikace. Pro tento účel byla zvolena databáze MongoDB[2]. Do ní lze ukládat data ve formátu JSON, který byl jako výstupním formátem zpracovaných dat v návrhu aplikace v kapitole 4.1.

[1] http://hadoopsdk.codeplex.com/
[2] http://www.mongodb.com/

5.1.1 Jazyk C#

Jedná se o vysokoúrovňový objektově orientovaný programovací jazyk určený pro platformu .NET Framework. Vyvíjí jej společnost Microsoft a je tedy určen pro platformu Windows. Syntaxí je podobný jazyku Java, se kterým sdílí některé koncepty. Jedná se se například o použití Garbage collectoru a základní třída od, které dědí všechny objekty. Garbage collector sleduje počty referencí na jednotlivá alokovaná paměťová místa, a pokud u některého z nich klesne počet referencí na nulovou hodnotu, tak může toto paměťové místo uvolnit. Druhá zmíněná společná vlastnost s jazykem Java se pro jazyky nad .NET Framework označuje jako Common Type System. Každý objekt je potomkem třídy `System.Object` a dědí od ní všechny její metody.

Při vytváření aplikace nad .NET Framework, což platí tedy i pro jazyk C#, hovoříme o tzv. assembly, která je po zkompilování reprezentována EXE souborem nebo DLL knihovnou. Tyto soubory nejsou ovšem zkompilované do nativního kódu procesoru, ale do CLI (Common Intermediate Language). CLI je množina instrukcí, která je nezávislá na procesoru a do nativního kódu procesoru je překládána v době spuštění, což bývá označováno jako JIT (just-in-time). Přesněji řečeno je každá metoda JIT kompilována když je zavolána. Při dalším volání se použije dříve provedený překlad a metoda není znova kompilována. O tuto kompilaci se stará překladač specifický pro danou architekturu, který je součástí vrstvy CLR (Common Language Runtime). CLR tedy řídí vykonávání vlastního programu a je to nejnižší vrstva v .NET Frameworku a tedy i nejblíže operačnímu systému.

5.1.2 Windows Presentation Foundation

Technologie Windows Presentation Foundation známá po zkratkou WPF představuje další generaci prezentačního systému pro vytváření aplikací na platformě Windows s vizuálně dokonalým uživatelským prostředím [12]. Vlastnosti, jenž tato technologie přináší, jsou následující [13]:

- Integrace mnoha technologií a jednotný programoví model.
- Nezávislost na rozlišení díky použití vektorové grafiky.
- Hardwarová akcelerace protože WPF je založeno na Direct3D.
- Deklarativní programování pro sestavení grafického rozhraní.
- Široké možnosti kompozice a úpravy grafických komponent.

Pro kompozici uživatelského rozhraní se používá deklarativní značkovací jazyk XAML (čti zaml) založený na značkovacím jazyce XML. Pokud vytváříme WPF aplikace s grafickým uživatelským rozhraním, tak máme kód pro grafické elementy (např. okno) rozdělen do dvou souborů. V jednom souboru definujeme právě pomocí jazyka XAML strukturu uživatelského rozhraní. Ve druhém potom máme tzv. code behind, kde zpracováváme zaregistrované události, předáváme data k zobrazení a implementujeme další aplikační logiku. Díky tomu lze při vývoji aplikací oddělit vývoj uživatelského rozhraní, který provádějí designéři, od vývoje aplikační logiky. Při takovémto odděleném vývoji uživatelské rozhraní neví, jaká data bude zobrazovat, ale zná pouze, jak je má zobrazit. Data se poskytují pomocí vazeb tzv. binding. Grafická komponenta poté data získaná přes vazbu vizualizuje, ale nereflektuje změny v datech změnou jejich zobrazení. To jí musí být explicitně sděleno, aby se vizualizace provedla znovu.

5.1.3 Microsoft .NET SDK For Hadoop

Jedná se o SDK (software development toolkit) sloužící pro podporu vývoje aplikací na platformě .NET pracujících s Apache Hadoop. Zejména je zaměřeno na spolupráci s Windows Azure HDInsight, což

je Apache Hadoop běžící jako služba v cloudu Windows Azure. Pro HDInsight nabízí podporu pro správu Hadoop clusteru a spouštění úloh tzv. jobů. Zahrnuje kromě podpory pro platformu .NET i cmdlety pro Windows PowerShell. Umožňuje provádět v .NET nebo v systému PowerShell MapReduce úlohy, přičemž může být využita databáze Apache Hive, úlohy napsané v jazyce Pig a streamované úlohy s využitím Hadoop Streaming..

Toto SDK zahrnuje mimo jiné i podporu pro vytváření aplikací, když Apache Hadoop neběží v cloudu Windows Azure, ale na běžné instalaci Apache Hadoop clusteru na platformě Window. Kromě podpory pro MapReduce úlohy, zahrnuje také rozhraní pro práci se souborovým systémem HDFS a podporu pro dotazovací jazyk LINQ nad databází Hive. Tato část projektu je zatím v experimentální fázi (květen 2014), a proto při použití nemusí být stabilní, na což by měl být brán zřetel.

5.1.4 Databáze MongoDB

V návrhu aplikace bylo známo, že použitá databáze pro ukládání zpracovaných dat nemůže mít pevné databázové schéma, protože u různých protokolů bude potřeba ukládat různé hodnoty. Využití relační databáze tedy nepřipadalo v úvahu a oblast zájmu se přesunula k NoSQL databázím. Dále bylo v návrhu počítáno také počítáno s tím, že výsledná aplikace by měla spolupracovat s Apache Hadoop lokálně nebo vzdáleně, tak by také měla využívat Windows Azure HDInsight. Ten ovšem nezahrnuje, jak je vidět na obrázku *Obrázek 3.7*, NoSQL databázi HBase, která je postavena nad souborovým systémem HDFS. Bylo se tedy nutné poohlédnout po jiné NoSQL databázi vhodné pro BigData. Nakonec byla zvolena databáze MongoDB, která tuto vlastnost splňuje a také existuje podpora pro jazyk C#.

MongoDB reprezentuje NoSQL databázový systém, jenž je dokumentově orientovaný. To znamená, že databáze má dynamické schéma a umožňuje ukládat různá data do jedné tabulky. V případě MongoDB hovoříme o tzv. kolekcích. Data (dokumenty) se do těchto kolekcí ukládají ve formátu BSON, který vychází z jednoduchého formátu JSON a rozšiřuje jej. Formát JSON ovšem ukládá data v textové podobě naproti tomu BSON ve formě binární. Proto také název BSON, který znamená „Binary JSON“.

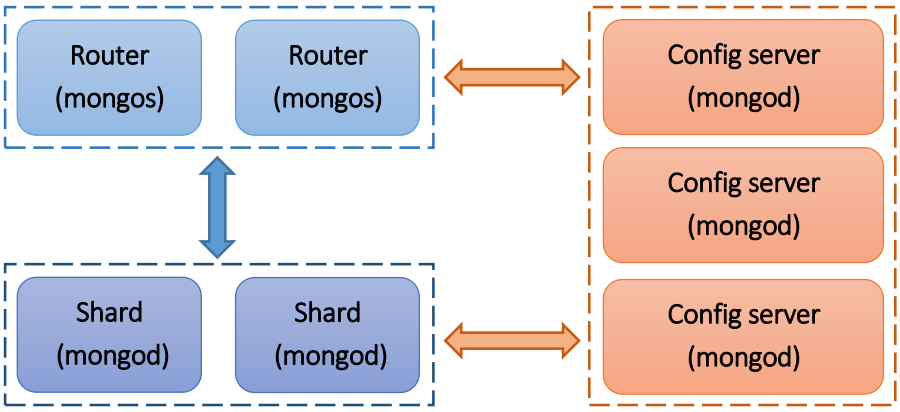
```
{
    "id": 1,
    "Jmeno": "Tomáš",
    "Prijmeni": "Baťa",
    "Narozeni": 1876,
    "Vlastnosti":["tvořivost","cílevědomost","mravní hodnoty"]
}
```

Ukázka dokumentu ve formátu JSON.

Tato databáze je hojně využívána, protože se jedná o open source projekt, který je multiplatformní. Je to projekt vcelku mladý, protože jeho první verze byla vydána teprve v roce 2009. Za tuto dobu však dokázal prorazit na poli databázových systémů, o čemž svědčí fakt, že je využíván i velkými společnostmi a webovými službami jako jsou Amazon Web Services, IBM či redhat [14].

Pro tento projekt byl zvolen tento databázový systém jednak dle podmínek uvedených dříve, ale také díky tomu, že je to databáze vhodná pro uložení velkých dat. Umožňuje totiž horizontální škálovatelnost [15] při zvětšování velikosti uložených dat. Při tomto jsou ukládaná data rozdělována mezi více serverů, které jsou na stejné úrovni. Této technice se v MongoDB prostředí říká *sharding*. Představuje proces rozdělení dat podle klíče mezi bloky zvané *shardy*. Jeden shard se skládá jednoho nebo více datových serverů. Tato skupina serverů se nazývá replikační množina (replica set), kde je jeden server master, který je určen pro čtení a zápisy a ostatní servery v replikační množině obsahují

kopii dat z master serveru a slouží pouze pro čtení. Druhou množinu serverů zastupují konfigurační servery (config servers). Tyto servery slouží pro uložení metadat. V produkčním režimu se nasazují tři servery, aby byla zajištěna redundance. Důležité je zmínit, že nepředstavují replikační množinu jako servery datové a tedy každý z nich musí být schopen provádět změny v metadatech. Nelze mít ovšem pouze jeden konfigurační server, protože ten by představoval, že v případě jeho nedostupnosti nebudou dostupná žádná data. Poslední komponentu v této technice představují směrovací servery (routing servers). Ty slouží jako přístupové body k databázi. Komunikují přitom z konfiguračními servery a získávají od nich metadata, která určují, kde jsou konkrétní data uložena. Odstiňují tedy klientskou aplikaci od celé vnitřní architektury, protože ta nikdy nepřistupuje přímo na shardy.



Obrázek 5.1: Konfigurace MongoDB clusteru se shardy

Na obrázku Obrázek *5.1* je ilustrována architektura databáze MongoDB při použití shardů, tak jak byla popsána. V závorce u každé komponenty je uveden název démona (služby), který je na daném serveru spuštěn a zajišťuje danou funkčnost. Z obrázku je patrné, že konfigurační server (config server) a shard obstarává stejný démon. V obou případech je ovšem spuštěn s jinými parametry.

5.1.4.1 MongoDB Connector For Hadoop

Jedná se o plugin pro Apache Hadoop framework. Ten umožňuje použít databázi MongoDB nebo soubory ve formátu BSON jednak jako zdroj dat do MapReduce úloh, tak také jako cílové úložiště pro výsledky MapReduce úloh. Celá implementace je samozřejmě v jazyku Java, protože Apache Hadoop je také napsán v jazyku Java. Kromě podpodpory MapReduce úloh v jazyce Java zahrnuje tento plugin také podporu pro skriptovací jazyk Pig a SQL jazyk Hive. Hadoop Streaming je také podporován [16], takže funkce Map a Reduce mohou být napsány v libovolném jazyce. V implementaci pro Hadoop Streaming je zahrnuta podpora pro vytváření MapReduce programů pro jazyky Ruby, Node.js a Python [16].

5.1.5 Infragistics

Aplikace s grafickým uživatelským rozhraním skládáme z grafických komponent. Tyto komponenty jsou většinou dostupné pro technologii (např. WPF) nad kterou grafickou aplikaci vyvíjíme. Takto bývají ovšem dostupné většinou pouze nějaké základní komponenty, které si lze sice stylovat do podoby jaké chceme dosáhnout, ale je to náročně na vývoj. Potenciál v této oblasti objevili některé společnosti. Ty nabízejí pro vývojáře aplikací různé balíčky grafických komponent. Jednou takovou společnost představuje Infragistics a její balíček Infragisics ULTIMATE, který zahrnuje velkou řadu různým grafických komponent pro různé platformy. Můžeme tedy říci, že se vývojáři aplikací zaměřují

spíše na funkční stránku aplikaci a grafické komponenty kupují. Tím se zlevní vývoj aplikace a zkrátí doba jeho vydání na trh.

Pro výslednou implementovanou aplikaci byl použit právě balíček komponent Infragisticts ULTIMATE. Ten zahrnuje komponenty pro desktop aplikace (Windows Form, WPF), mobilní platformy (Windows Phone, iOS a Android) a webové platformy (ASP.NET, Silverlight a JQuery/HTML5). Jedná se tedy o velice komplexní balíček. Z něhož jsme ve výsledné aplikaci použili komponenty pro tabulky s možností filtrování a grafy zobrazující statistiky, ze zpracovaných dat. Zejména v případě grafů by vytvoření jejich komponent nebylo jednoduché, a proto byly použity z tohoto balíčku.

5.2 Struktura aplikace

Aplikace byla rozdělena z důvodů znovu použitelnosti rozdělena do více projektů. Tímto rozdělení dosáhneme možné znovu použitelnosti částí řešení, protože se ve výsledku nebude jednat pouze o jeden EXE soubor. Rozdělení bylo provedeno, tak aby spolu logicky spojené části, které tvoří funkční celek, patřili do jednoho projektu, na který bude se v hlavní aplikaci odkazovat. Takovýto funkční celek bude poté možné využít v dalších vyvíjených aplikacích.

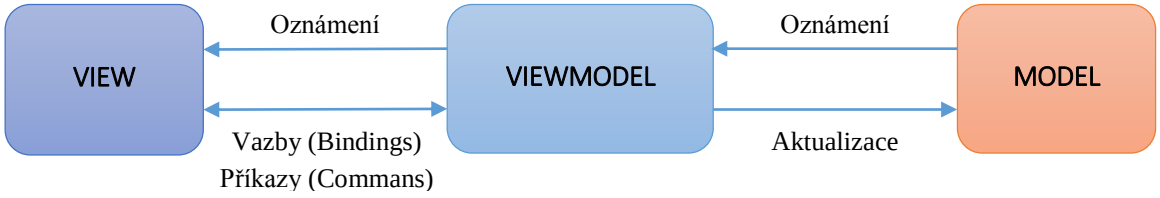
Výsledná aplikace byla rozdělena do projektů, které jsou popsány v tabulce *Tabulka 5.1*. Jeden projekt je na výstup reprezentován jednou assembly (spustitelný EXE soubor nebo DLL knihovna).

Jmenný prostor	**Popis**
NetfloxMontana	Hlavní aplikace s grafickým uživatelským rozhraním.
NetfloxMontana.Common	Třídy řešící obecnou problematiku při vývoji aplikací.
NetfloxMontana.MapReduce	Zpracování MapReduce úloh na Apache Hadoop.
NetfloxMontana.Parse	Parsery aplikačních protokolů.

Tabulka 5.1: Rozdělení do projektů

5.2.1 Návrhový vzor Model View ViewModel

Model View ViewModel zkráceně MVVM představuje návrhový vzor pro aplikace s grafickým uživatelským rozhraním, který byl použit jako základní koncept při vývoji výsledné WPF aplikace. Jedná se o koncepci, ve které je grafické uživatelské rozhraní (prezentační logika) odděleno od aplikační logiky. Tím je dosaženo lepší udržovatelnosti aplikace a možné rozdělení práce v týmu. Lepší udržovatelnost lze zajistit díky onomu oddělení, protože lze jednotlivé části nezávisle testovat přes dohodnuté rozhraní. Tohoto rozhraní se právě využívá i při práci v týmu, kdy je dohodnuto toto rozhraní a části aplikace se mohou vyvíjet nezávisle a až po otestování se spojí do funkčního celku. Rozeznáváme tři entity, které spolu komunikují, jak je ukázáno na obrázku *Obrázek 5.2*. Přičemž každá entita je zodpovědná za jinou část funkčnosti aplikace.



Obrázek 5.2: Návrhový vzor Model View ViewModel

První entita View (pohled) zapouzdřuje grafické uživatelské rozhraní. Její odpovědností je definovat strukturu a vzhled toho, co uživatel vidí na obrazovce [17]. Definuje tedy grafické komponenty a jejich umístění. Data z modelu, která se budou zobrazovat, jsou navázána pomocí vazeb tvz. bindings ze druhé entity ViewModel. Pohled pouze vykreslí do grafických komponent data, která mu dodal ViewModel. Pokud tedy nastane aktualizace dat v modelu a chceme tuto změnu vizualizovat v pohledu, tak je to nutné pohledu oznámit. K tomuto účelu slouží oznámení neboli notifikace, pomocí kterých říkáme pohledu, že došlo ke změně dat a měl by zobrazit aktuální data.

Druhou entitou, která již byla částečně zmíněna v předešlém odstavci v návaznosti na fungování pohledu je ViewModel. Jeho úkolem zajišťovat prezentační logiku aplikace. Nemá přitom žádnou přímou referenci na pohled ani žádnou informaci o specifické implementaci modelu [17]. Slouží jako prostředník mezi pohledem a modelem. Poskytuje data pohledu, zasílá mu oznámení o změně dat a v neposlední řadě definuje příkazy. Příkazy se stejně jako data navazují na grafické komponenty např. tlačítko. Pokud dojde poté ke kliknutí na toto tlačítko, vyvolá se daný příkaz definovaný ve ViewModelu. Vyvoláním se rozumí provedení obsluhy daného příkazu.

Poslední částí je model, jehož vybraná data poskytuje ViewModel pro pohled. Pokud se tato data aktualizují v pohledu, tak se díky vazbě aktualizují i modelu přes mezi článek ViewModel. Přitom může ViewModel kontrolovat, jestli data odpovídají požadovaným omezením a příp. může tuto aktualizaci eliminovat a vyvolat v pohledu oznámení pro uživatele. Z dosavadního výkladu se může zdá, že model poskytuje pouze data pro pohled. Ten ovšem můžeme implementovat další funkce pro práci s daty a tedy v objektově orientovaném vývoji může být reprezentován libovolným objektem.

5.2.2 Nastavení aplikace

Jednou z částí implementované aplikace je možnost nastavení parametrů pro připojení ke clusteru Apache Hadoop a pro připojení databázi MongoDB. To přináší výhodu v podobě možného využití různých výpočetních clusterů pouhou změnou konfigurace. V případě databáze MongoDB si můžeme vytvořit několik oddělených databázi a používat je pro ukládání výsledků zpracování PCAP souborů, které pochází například z různých sítí a tím si oddělit celkové statistiky jednotlivých sítí. Výhodou databáze MongoDB je, že pokud má připojující se uživatel přidělena uživatelská pro vytváření databáze a databázových kolekcí, tak se nové databáze a databázové kolekce vytvoří při připojení resp. při prvotním ukládání do příslušné kolekce. Za tuto vlastnost vděčíme dynamickému databázové schématu.

5.3 Hadoop streaming zpracování

Jak je uvedeno v kapitole 3.3.3, tak pro spouštění MapReduce úloh na Apache Hadoop, kde mapper a reducer jsou libovolné spustitelné programy, využíváme Hadoop Streaming. Tento způsob zpracování byl využit i pro zpracování PCAP souboru, protože jako implementační jazyk byl zvolen C#. Využívá se přitom balíku Microsoft .NET SDK For Hadoop. Ten obsahuje již spustitelné EXE programy pro mapper a reducer. Tyto programy ovšem sami o sobě neprovádí žádnou MapReduce úlohy, ale pouze řídí její provádění dle předané konfigurace. Řízení spočívá ve vytvoření instance třídy, která implementuje metodu Map resp. Reduce a následném čtení dat po řádcích ze standardního vstupu, předávání těchto jednotlivých řádků do metody Map resp. Reduce. Zasílání výsledků na standardní výstup ovšem nezajišťuje. Tato činnost se provádí přes kontext zpracování fáze mapping resp. reducing, který je vytvořen v mapperu resp. reduceru. Právě textové zpracování po řádcích se zdálo zpočátku limitující, protože formát PCAP je ukládá data v binární podobě. Bylo ovšem nalezeno řešení, kterému

toto zpracování vstupu a výstupu po řádcích v mapper a reduceru nevadí. Implementace tohoto řešení je popsána v kapitole 5.4.1.

Řekli jsme, že mapper a reducer neprovádí MapReduce úlohu, ale pouze vytvoří instanci třídy pro mapping resp. reducing a řídí její provádění a předávání vstupu do dané fáze. Třídy provádějící fázi mapping a reducing musí tedy být určitého datového typu, aby byl schopen mapper resp. reducer provést jejich instanci, a poté s nimi komunikovat přes dohodnuté rozhraním. V případě že se jedná mapper, tak musí třída implementující fázi mapping být potomkem třídy `MapperBase` ze jmenného prostoru `Microsoft.Hadoop.MapReduce` a implementovat abstraktní metodu `Map`. Mapper poté volá tuto metodu Map pokaždé, když přečte ze standardního vstupu jeden řádek dat, a poté je již na konkrétní implementaci, jakým způsobem budou data zpracována. Druhá část zpracování reducer pracuje stejným způsobem jako mapper. Třída provádějící fázi reducing musí být ale potomkem třídy `ReducerCombinerBase`, která je ve stejném jmenném prostoru jako třída `MapperBase`. Zde se musí implementovat abstraktní metoda `Reduce`, která přijme klíč a seznam hodnot obsahující hodnoty, které byly s tímto klíčem odeslány z fáze mapping. Výstup z fáze reducing je řešen stejně jako u fáze mapping přes zápis do kontextu zpracování, který data zapíše na standardní výstup.

Specifické třídy provádějící fáze mapping a reducing se předávají do Hadoop Streaming přes parametr `-cmdenv name=value`. Tento parametr se definuje celkem čtyřikrát s hodnotami name a value uvedenými v tabulce *Tabulka 5.2*.

Název (name)	**Hodnota (value)**
MSFT_HADOOP_MAPPER_DLL	DLL knihovna obsahující třídu uvedenou v parametru MSFT_HADOOP_MAPPER_TYPE.
MSFT_HADOOP_MAPPER_TYPE	Třída provádějící fázi mapping.
MSFT_HADOOP_REDUCER_DLL	DLL knihovna obsahující třídu uvedenou v parametru MSFT_HADOOP_REDUCER_TYPE.
MSFT_HADOOP_REDUCER_TYPE	Třída provádějící fázi reducing.

Tabulka 5.2: Parametry pro definici tříd provádějící fáze mapping a reducing.

5.3.1 Konfigurace Hadoop Streaming pro zpracování PCAP souboru

Program Hadoop Streaming potřebuje pro úspěšné spuštění MapReduce úlohy minimálně čtyři parametry uvedené v tabulce *Tabulka 3.2*. Pro potřeby zpracování PCAP souboru musíme zadat mnohem více parametrů, protože výstup z fáze reducing budeme zapisovat do databáze MongoDB a využíváme také balíček Microsoft .NET SDK For Hadoop. Vlastní příkaz pro spuštění provádění streamované úlohy bude následující.

```
$HADOOP_HOME\bin\hadoop.cmd jar $HADOOP_HOME\lib\hadoop-streaming.jar
    -D "tmpjars=<JAR archivy potřebné pro outputformat>"
    -D "mongo.output.uri=<URL pro připojení k databázi MongoDB>"
    -D "cz.vutbr.fit.netfloxmontana.jobid=<ID úlohy>"
    -D "mapred.map.max.attempts=1"
    -D "mapred.reduce.max.attempts=1"
    -files <seznam potřebných souborů pro mapper a reducer>
    -input hdfs:///jobs/<ID úlohy>/<vstupní soubor>
    -output hdfs:///jobs/<ID úlohy>/out
    -outputformat cz.vutbr.fit.netfloxmontana.MongoOutputFormat
    -mapper Microsoft.Hadoop.MapDriver.exe
```

```
-reducer Microsoft.Hadoop.ReduceDriver.exe
-cmdenv "MSFT_HADOOP_REDUCER_DLL=NetfloxMontana.MapReduce.dll"
-cmdenv "MSFT_HADOOP_REDUCER_TYPE=NetfloxMontana.MapReduce.FlowReducer"
-cmdenv "MSFT_HADOOP_MAPPER_DLL=NetfloxMontana.MapReduce.dll"
-cmdenv "MSFT_HADOOP_MAPPER_TYPE=NetfloxMontana.MapReduce.FlowMapper"
```

Význam generický parametrů pro implementovanou aplikaci:

Parametr	**Význam**
-D tmpjars	Java JAR archivy související s implementací třídy výstupního formátu v parametru -outputformat.
-D mongo.output.uri	URI pro připojení k databázi pro Java třídu výstupního formátu uvedeného v parametru -outputformat.
-D cz.vutbr.fit.netfloxmontana.jobid	Identifikátor úlohy pro označení příslušnosti datových toků k dané úloze.
-D mapred.map.max.attempts	Maximální počet pokusů provádění fáze mapping.
-D mapred.reduce.max.attempts	Maximální počet pokusů provádění fáze reducing.
-files	Seznam souborů využívaných pro fáze mapping nebo reducing mimo Java JAR archivy.

Význam Streaming parametrů:

Parametr	**Význam**
-input	Vstupní soubor do MapReduce úlohy.
-output	Výstupní složka pro MapReduce úlohu.
-mapper	Spustitelný program z balíku Microsoft .NET SDK For Hadoop provádějící fázi mapping.
-reducer	Spustitelný program z balíku Microsoft .NET SDK For Hadoop provádějící fázi reducing.
-outputformat	Třída výstupního formátu provádějící ukládání výsledků z fáze reducing do databáze MongoDB
-cmdenv	Význam uvádí tabulka Tabulka *5.2*

5.3.2 Lokální zpracování

Pokud budeme spouštět výslednou aplikaci přímo na serveru, na němž máme nainstalován Apache Hadoop, tak můžeme použít lokální připojení na Hadoop cluster. Aplikace se stejně může připojovat přes vzdálené připojení po sítí, které se ovšem změní na komunikaci přes lokální síťovou smyčku tzv. loopback. Při nastavení lokálního připojeni se ve skutečnosti ani připojením nejedení, ale z důvodů jednotné terminologie jej, tak nazýváme. Pokud totiž spouštíme program přímo na serveru, tak máme přístup k programu pro ovládání souborového systému HDFS a programu pro Hadoop Streaming. Chceme-li tedy provést požadovanou akci, což může být např. zkopírování souboru do HDFS nebo spuštění streamované úlohy přes Hadoop streaming, tak programově spustíme konkrétní program. Předáme mu přitom takové parametry, aby provedl požadovanou akci, a dále čekáme na jeho dokončení, abychom získali výsledek operace. Z výstupu poté zjistíme, jestli byla operace úspěšná nebo neúspěšná. Při lokálním připojení tedy neprobíhá žádní komunikace přes síťové rozhraní, ale spouští se přímo procesy obslužných programů.

Každé připojení ať už se jedná o lokální nebo vzdálené obsahuje dvě rozhraní. První rozhraní pro přístup k souborovému systému HDFS a druhé pro konfiguraci a spouštění streamovaných

MapReduce úloh přes Hadoop streaming. Implementace obou těchto rozhraní je vytvořena v balíku Microsft .NET SDK For Hadoop. Ve výsledné aplikaci se ovšem používá z tohoto balíku pouze rozhraní pro přístup k souborovému systému HDFS. Druhé rozhraní pro spouštění MapReduce streamovaných úlohy muselo být rozšířeno a celé implementováno, protože implementace v balíku Microsoft .NET SDK For Hadoop neumožňovala nakonfigurovat Hadoop streaming dle potřeb pro zpracování PCAP souboru. Implementace v tomto balíku totiž nezahrnuje všechny potřebné parametry. Dalším problém, který potřeboval k vyřešení vytvořit vlastní implementaci pro spouštění streamovaných úloh, nastával s parametrem *-files* v konfiguraci Hadoop streaming. Implementace v balíku Microsft .NET SDK For Hadoop totiž do tohoto parametru uvádí všechny DLL knihovny včetně vlastní grafické EXE aplikace, jenž tato aplikace používá. Poté nastává problém, že výsledný příkaz pro spuštění Hadoop streaming s parametry uvedenými v kapitole 5.3.1 přesáhne maximální délku příkazu, kterou umožňuje příkazový řádek v systému Windows spustit. Je to dáno jednak tím, že se do HDFS kopírují zbytečné knihovny, které nejsou vůbec pro vykonávání MapReduce úlohy potřeba. Za druhé se jednotlivé cesty k uváděným souborů uvádí jako plně kvalifikované ve formě URL. Nejedná se tedy pouze o názvy souborů oddělených čárkami, které jsou výrazně kratší než plně kvalifikované cesty ve formě URL. Tento druhý problém není až tak velkou překážkou, pokud se budou do HDFS nahrávat pouze knihovny potřebné pro programy provádějící MapReduce zpracování. O tomto možném omezení se v počátcích práce, když se ověřovalo možné využití Apache Hadoop pro zpracování PCAP souboru nevědělo a vyústilo v omezení funkčnosti, než bylo plánováno v návrhu aplikace.

V návrhu v kapitole 4.1 bylo plánováno, že parsery aplikačních protokolů se budou do aplikace přidávat formou pluginů a aplikace bude dobře rozšiřitelná na zpracování dalších protokolů. Po objevení omezené v délce příkazu by se totiž může lehce stát, že příkaz pro Hadoop Streaming přesáhne maximální povolenou délku a MapReduce úlohy poté nepůjdou vůbec spustit. Rozhodl jsem se tedy všechny implementace aplikační parserů umístit do jedné DLL knihovny, kterou reprezentuje projekt ve jmenném prostoru NetfloxMontana.Parse.

Dalším vylepšením, které podpořilo zrychlení vytvoření MapReduce úlohy bylo nenahrávání DLL knihoven uvedených v parametru *-files* pro každou úlohu, jak je to uděláno v balíku Microsoft .NET SDK For Hadoop. Ve vytvořené implementaci se tyto soubory nahrávají pouze při vytváření první úlohy, a poté se již pouze kontroluje, jestli některý soubor nechybí. Nahrávání se provádí do složky */libs* umístěné v kořenu HDFS, aby byly poté kvalifikované cesty souborů v parametru *-files*, co nejkratší a výsledný příkaz nepřesáhl povolenou délku.

Ve vlastní implementaci se o toto lokální zpracování starají tři třídy ze jmenného prostoru `NeftloxMontana.MapReduce.Executuion`. Rozdělení je provedeno záměrně, abychom se v aplikaci abstrahovali od streamovaných úloh a jejich řízení. Toto rozdělení je zajištěno v implementaci konkrétního připojení k Hadoop clusteru, které zajistí vytvoření správné instance vykonavatele tzv. executor. Tento vykonavatel zahrnuje rozhraní pro spouštění MapReduce úloh. V první fázi tedy pracuje s MapReduce úlohou a nestavíme se k ní jako ke streamované úloze. To zajišťuje až vykonavatel úlohy, který konfiguruje lokálního vykonavatele streamované úlohy tzv. streaming executor. Zde se v případě lokální úlohy vytvoří vykonavatel procesu tzv. process executor, který spustí dle obdržené konfigurace proces pro Hadoop streaming a předává informace z jeho standardního výstupu a standardního chybového výstupu vykonavateli streamovavné úlohy. Ten tyto informace dále deleguje, čímž je možné je zobrazovat v aplikaci.

5.3.3 Vzdálené zpracování

V případě vzdáleného zpracování se k Hadoop clusteru připojujeme přes počítačovou sít. Nelze proto zpracování provádět stejně jako v případě lokálního, protože není možné přímo na serveru spouštět příkazy pro ovládání souborového systému a aplikaci pro Hadoop streaming pro vykonávání steamovaných úloh. Místo toho se komunikuje se severem přes protokolem HTTP přes dvě REST[1] API[2]. REST označení znamená, že se jedná o datově orientované rozhraní, které je bezstavové. Pro identifikaci stavu aplikace a jejího chování slouží identifikátor, který musí být unikátní pro každý jedinečný zdroj. Ten je zastoupen v HTTP protokolu při požadavku na server jako požadované URI tzv. request URI. Dále tento přístup definuje jednotné rozhraní pro manipulaci s požadovaným zdrojem uvedeným v URI. Používá model čtyř operací označovaný pod zkratkou CRUD (create, read, update a delete). Tyto operace jsou poté mapovány na dotazovací metody protokolu HTTP (create – POST/PUT, read - GET, update – PUT, delete – DELETE).

Pro práci se souborovým systémem HDFS se využívá WebHDFS REST API. Pro spouštění MapReduce úloh zase slouží WebHCat REST API. V případě obou uvedených rozhraní se přenáší data ve formátu JSON, což umožňuje jejich jednoduché zpracování. Při použití vzdáleného zpracování se ve výsledku provádí stejném příkazy jako při lokálním zpracování. Rozdíl je v tom, že při vzdáleném zpracování nemusí aplikace využívající Apache Hadoop být spuštěna přímo na serveru obsahující instalaci toho frameworku.

Vlastní implementace vzdáleného zpracování je stejně jako v případě lokálního zpracování zahrnuta do jmenného prostoru `NeftloxMontana.MapReduce.Executuion`. O provádění se tentokrát starají pouze dvě třídy, protože není nutné zajišťovat správu procesu provádějící Hadoop streaming. První třída `WebJobExecutor` slouží pro abstrakci od streamované úlohy stejně jako v případě lokálního zpracování a provádí konfiguraci vykonavatele vzdálené streamované úlohy. Jehož funkčnost implementuje třída `WebStreamingExecutor`, které přes WebHCat REST API vytvoří novou streamovanou úlohu a poté se dotazuje na její stav a ten předává dále pro další zpracování.

5.3.3.1 WebHDFS REST API

Ve výsledné aplikaci využíváme tohoto rozhraní pro operace na Hadoop souborovým systémem nad HDFS. Vlastní implementace klienta pro toto rozhraní nebyla součástí tohoto projektu, ale využívá se implementace z balíku Microsoft .NET SDK For Hadoop. Ta poskytuje rozhraní pro práci nad HDFS v aplikacím psaných v jazyce C#.

5.3.3.2 WebHCat REST API

WebHCat představuje rozhraní pro přístup ke komponentě Apache HCatalog, dříve známé pod názvem Templeton, která je zahrnuta v Apache Hadoop frameworku. Tato komponenta představuje abstraktní vrstvu, která umožňuje zpracovávat data různými nástroji, jako jsou Apache Pig, Apache MapReduce a Apache Hive [18]. Pro externí aplikace poskytuje právě toto REST rozhraní.

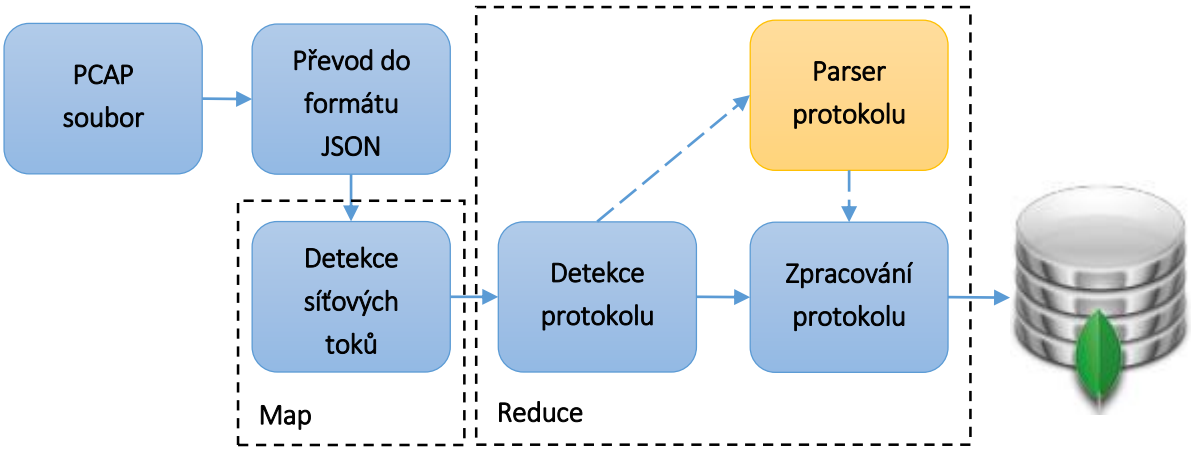
V implementované aplikaci využíváme tohoto rozhraní pro vytváření a spouštění streamovaných MapReduce úloh a k následnému dotazování se na jejich stav. Nevýhodou tohoto rozhraní je, že pro streamované MapReduce úlohy, nepodporuje všechny parametry, které poskytuje Hadoop Steaming. V případě zpracování PCAP souboru potřebujeme předat v generickém parametru `-libjars` seznam Java JAR archivů souvisejících s implementací výstupního formátu. Rozhraní sice nabízí možnost

[1] Representational State Transfer
[2] Application Programming Interface

předat v parametru `arg` další parametry. Tyto parametry se ovšem při vytváření příkazu pro spuštění Hadoop Steeaming předají do streamovacích parametrů (streamingOptions), ale parametr `-libjars` spadá do generických parametrů (genericOptions). V tomto případě tedy nedojde vůbec ke spuštění Hadoop Streaming. Řešení ovšem nabízí parametr `define` pro předávání Hadoop konfiguračních proměnných, které se objeví v příkazu pro Hadoop Streaming jako řetězce ve formátu název=hodnota v parametrech `-D`.

5.4 MapReduce zpracování PCAP souboru

Náplní této kapitoly je objasnit, jaké bylo navrženo a implementováno zpracování PCAP souboru se zachycenou síťovou komunikací přes programovací model MapReduce na Apache Hadoop clusteru s využitím Hadoop streaming. Bude objasněno, co se prování v jednotlivých fázích zpracování. Jaké datové jednotky se přenáší mezi jednotlivými fázemi. Prostě kompletní navržené a implementované řešení, se zaměřením na klíčové části, jenž udávali směr výsledného řešení. Celé řešení zpracování, kdy na vstupu máme PCAP soubor se zachycenou síťovou komunikací a výsledek zpracování toho souboru ukládáme do databáze MongoDB ilustruje obrázek *Obrázek 5.3*. V dalších částech této kapitoly budou následovat rozebrány jednotlivé části tohoto zpracování.



Obrázek 5.3: Schéma zpracování PCAP souboru

5.4.1 Převod do formátu JSON

Zpracování přes model MapReduce začíná ve frameworku Apache Hadoop čtením vstupních dat a jejich následným předáváním do fáze mapping. To jaká data budou vstupu do tohoto procesu je určeno v konfiguraci MapReduce úlohy. Stejně tak musí v této konfigurací uvedeno, jak se budou tato data číst a jaké datové jednotky budou předávány na vstup fáze mapping. Tuto funkčnost zajišťuje vstupní formát, který odstiňuje MapReduce zpracování od zdroje dat, z nějž se budou data zpracovávat. Nemusí se tedy vždy jednat pouze o datové soubory. Vstup se můžeme zpracovávat tedy i z databáze nebo přímo od různých senzorů, přitom záleží pouze na implementaci vstupního formátu.

Vstupní formát představuje v Apache Hadoop frameworku třída napsaná v jazyku Java, která implementuje rozhraní `org.apache.hadoop.mapred.InputFormat<K,V>`. Toto rozhraní zahrnuje pouze dvě metody. První metoda `getSplits` by měla rozdělit vstupní data do logických částí, pro vstup do jednotlivých Map úloh spuštěných ve fázi mapping MapReduce zpracování. Toto rozdělení tedy v MapReduce zpracování reprezentuje fázi partitioning. Druhá metoda `getRecordReader` by měla implementovat vytvoření instance čtečky záznamů, která implementuje

rozhraní `org.apache.hadoop.mapred.RecordReader<K,V>`. Jedna instance této čtečky záznamů neboli record readeru zpracovává jeden logický celek tzv. input split, které vytvořila metoda getSplits vstupního formátu. Slouží tedy jako dodavatel dat do jednotlivých Map úloh ve fázi mapping a na její implementaci spočívá možnost číst daný formát. Instalace Apache Hadoop frameworku zahrnuje několik implementovaných vstupních formátů. Jsou zde obsaženy jednak formáty pro textové a binární soubory, kdy lze mít na vstupu jeden nebo více vstupních souborů, tak i třeba vstupní formát pro čtení vstupu z relační databáze. V obou případech uvedených rozhraní se jedná o generické rozhraní, kde generický typ `K` představuje datový typ klíče a generický typ `V` datový typ hodnoty. Tyto datové typy musí implementovat příslušný mapper ve své metodě `Map`.

Na obrázku Obrázek *3.5* je zobrazena typická konfigurace Hadoop clusteru, kdy na jednom podřízeném (slave) uzlu běží démon TaskTracker pro vykonávání Map a Reduce a druhý démon DataNode obsluhující část dat souborového systému HDFS. Jak bylo uvedeno v předchozím odstavci, tak zpracování začíná rozdělení vstupu na jednotlivé logické celky ve fázi partitioning a jejich následným čtením do příslušných fází Map. Při rozdělování souboru na logické části bychom měli respektovat jednotlivé bloky souboru v souborovém systému HDFS. Rozdělení souboru by se tedy mělo řídit v první fází velikostí bloku v tomto souborovém systému, aby mapper spuštěný na daném uzlu přijímal data uložená a na témže uzlu a zamezilo se datovým přenosům mezi jednotlivými uzly po propojovací síti. Datové přenosy po propojovací síti se snažíme zamezit, protože čekání na data snižuje výkonost. Zamezit těmto datový přenosům úplně nelze nikdy, protože začátek logického celku musí začínat na začátku jednoho menšího logického celku a stejně tak konec logického celku musí být zarovnán na konec tohoto menší celku. Tento menší celek může přestavovat třeba jeden řádek textu ve vstupní souboru, a tedy v daném případě je logický celek reprezentován množinou řádků textu. Takový řádek textu nemusí, ale končit přesně s koncem logického celku a může zasahovat do dalšího bloku v souborovém systému HDFS, který může být uložen třeba na jiném uzlu. Tento přesah se tedy musí přenášet přes propojovací síť a není možné zamezit datovým přenosům mezi uzly, ale pouze je omezit. V případě zpracovávání po řádcích můžou být datové přenosy různé velikosti, protože předem nevíme, jaká velikost do konce řádku bude přesahovat.

Struktura formátu PCAP se skládá z několika polí, které byly popsány v kapitole 2.2.1.2. Pole Global Header a Packet Header mají pevnou délku. Třetí pole Packet Data má variabilní délku, která je uvedena v poli Packet Header. Abychom mohli z tohoto specifického formátu číst data a předávat je na vstup fáze mapping, tak bychom museli vytvořit vstupní formát pro PCAP soubor s příslušnou čtečkou záznamů. Vytvoření čtečky by nepředstavovalo problém. Ten se skrývá v implementaci druhé metody vstupního formátu getSplits zajišťující rozdělení na logické celky. Chtěli bychom začátek logického celku mít zarovnán na začátek pole Packet Header a konec logického celku ukončit s koncem paketu, tedy s polem Packet Data. Toto rozdělení nejsme schopni zajistit, protože velikost pole Packet Data je variabilní. Problémem je, že pokud se umístíme na nějakou pozici v souboru kopírující velikost bloku, tak nejsme schopni určit začátek a konec logického bloku. Řešením by bylo, kdyby začátek pole Packet Header šlo jednoznačně určit, což ovšem není možné. Poté zbývá možnost implementovat vstupní formát, kde logický blok bude shodný s velikostí celého souboru. To ovšem není efektivní implementace, protože by se spustilo provádění fáze mapping pouze na jednom uzlu. Zpracování by tedy probíhalo sekvenčně a vůbec by nebyla využita paralelizace.

Problém rozdělení souboru ve formátu PCAP na logické celky vyústil v rozhodnutí převést PCAP soubor před prováděním MapReduce úlohy do jiného formátu. Jako vhodný formát bylo zvolen formát JSON, kdy na jednom řádku ve vstupním souboru budeme mít jeden serializovaný rámec. Zápis ve formátu JSON pro jeden paket má následující strukturu.

```
{
    "Captured":<čas zachycení paketu>,
    "LinkLayer":<identifikátor linkové vrstvy>,
    "Data":<data rámce>
}
```

Struktura uvedená výše zahrnující pouze tři hodnoty je dostačující. Hlavně budeme potřebovat ve fázi mapping rekonstruovat zachycený síťový rámec do datové struktury, abychom mohli identifikovat síťový tok. K této rekonstrukci potřebujeme znám typ linkové vrstvy pro určení hlavičky rámce.

Převodem do JSON formátu se vyřešili dva problémy. Za prvé můžeme díky ukládání jednoho paketu na jeden řádek pracovat se vstupním souborem jako s textovým po řádcích. Díky tomu můžeme využít výchozí vstupní formát, který používá Hadoop streaming. Tento formát umí rozdělit vstupní soubor na logické celky a z nich implementovaná čtečka záznamu čte jednotlivé řádky. Dosáhneme tím tedy požadované paralelizace ve fázi mapping. Druhý problém, který se vyřešil, souvisí s použitím spustitelným programů pro mapper a reducer z balíku Microsoft .NET SDK For Hadoop. Tyto programy jsou totiž implementovány na vstupu a výstupu pro práci s řádky textu.

Vlastní převod do formátu JSON se provádí pomocí knihovny SharpPcap[1], která využívá knihovnu WinPcap, jejíž instalace je vyžadována. Knihovna SharpPcap slouží jednak pro zachytávání síťové komunikace ze síťových rozhraní, tak také pro čtení již zachycených dat ve formátu PCAP. Při převodu se využívá serializace objektů. Převod tedy spočívá v přečtení jednoho paketu ze vstupního souboru, naplnění příslušného objektu a jeho serializace do formátu JSON a následný zápis do souboru.

5.4.2 Detekce síťových – fáze mapping

Detekce síťových toků představuje první činnost za účelem získání dat ze síťového toku. Tento proces se provádí ve fázi mapping zpracování MapReduce úlohy. Úkolem této fáze je ze vstupního paketu identifikovat klíč síťového toku skládajícího se z pětice hodnot zahrnující

- zdrojová IP adresa,
- cílová IP adresa,
- zdrojový port,
- cílový port,
- transportní protokol.

Takovýto klíč bude identifikovat v dalších fázích zpracování příslušná data spadající do daného síťového toku. Výstupem této fáze je tedy klíč a k němu příslušná data z paketu. Tato dvojice vstupuje do další fáze MapReduce úlohy zvané shuffling. V ní se data z příslušným klíčem seskupí, a poté jsou předávány do další fáze reducing. Tím dosáhneme toho, že zprávy spadající do jednoho síťového toku jsou seskupeny, a poté jeden reducer ve fázi reducing zpracovává několik po sobě následujících síťových toků.

Implementaci fáze mapping představuje ve výsledné aplikací třída `FlowMapper` ze jmenného prostoru `NetfloxMontana.MapReduce`. Funkce je taková, že se nejprve rekonstruuje paket z bajtů dat a typu linkové vrstvy do datové struktury. K tomu se využívá knihovna Packet.Net[2]. Rekonstruovaný rámec linkové vrstvy se dále rozbaluje a získávají se datové jednotky na vyšších vrstvách komunikačního modelu TCP/IP. To znamená, že na vyšší vrstvě očekáváme IP datagram. Ten obsahuje zdrojovou a cílovou IP čímž získáme první dvě položky pro klíč síťového toku. V další fázi

[1] http://sourceforge.net/projects/sharppcap/
[2] http://sourceforge.net/projects/packetnet/

zjišťujeme, jaký byl použit protokol na transportní vrstvě. Z protokolů transportní vrstvy zpracováváme protokol TCP a UDP, které obsahují zdrojový a cílový komunikační port. Pokud se nejedná o komunikaci na transportní vrstvě, tak zpracováváme ještě protokol ICMP. Místo zdrojového a cílového portu uvádíme v tomto případě identifikátor ICMP relace. Tím jsme získali kompletní klíč identifikující síťový tok a dále stačí předat data do další fáze MapReduce zpracování. Dále předáváme klíč identifikující síťový tok a data z aplikační vrstvy (v případě protokolu ICMP se jedná o ICMP paket) a čas zachycení těchto dat, který budeme využívat ve fázi reducing.

5.4.3 Detekce a zpracování aplikačního protokolu – fáze reducing

Fáze reducing dostává na vstup seznam hodnot, které byly ve fázi mapping označeny stejným klíčem. O rozdělení hodnot do správného seznamu se stará fáze shuffling, která se provádí mezi fázemi mapping a reducing. Díky tomu zpracovává jedno volání metody Reduce seznam hodnot označených daným klíčem.

Vlastní detekce aplikačního protokolu ve fázi reducing je implementována ve třídě `FlowReducer`, která je umístěna ve jmenném prostoru `NetfloxMontana.MapReduce`. Určení protokolu je založeno na základě čísla známého portu, na kterém se komunikuje. Tímto určením si vytvoříme instanci příslušného parseru pro daný protokolu, kterým budeme postupně zpracovávat všechny zprávy označené daným klíčem. Jelikož výběr parseru závisí na čísle portu, který je součástí klíče, jímž označená dat zpracováváme v jednom volání metody Reduce tak na všechny zprávy použijeme jeden stejný parser. Úlohou parseru je získat z dané zprávy vybrané data, jejichž výběr závisí na implementaci daného parseru. V aplikaci jsou implementovány parsery pro protokoly HTTP, FTP, SMTP a ICMPv4.

Kromě vlastního parsování protokolu a získání požadovaných dat z něj, se v metodě Reduce provádí ještě určení začátku a konce datového toku. Tyto dvě hodnoty ohraničující začátek a konec komunikace se získají z času zachycení jednotlivých paketů, který je uložen ve formátu PCAP v poli Packet Header. Jeho hodnotu jsme předávali společně s daty dané zprávy z fáze mapping.

5.4.4 Datová komunikace během MapReduce zpracování

Během zpracování Hadoop streaming se komunikuje během mezi jednotlivými fázemi MapReduce zpracování streamované úlohy přes standardní výstup a standardní vstup. Na standardní výstup jsou zapisována data v jedné fázi a v následující jsou čtena ze standardního vstupu.

Jak ukazuje obrázek Obrázek *3.6*, tak zpracování začíná čtením zdrojových dat z určeného zdroje danou čtečkou záznamu vytvořenou ve zvoleném vstupním formátu. V našem případě při zpracování PCAP souboru máme na vstupu soubor převedený do formátu JSON, který obsahuje na jednom řádku záznam se strukturou.

```
{
    "Captured":<čas zachycení paketu>,
    "LinkLayer":<identifikátor linkové vrstvy>,
    "Data":<data rámce>
}
```

Tyto záznamy jsou čteny po jednotlivých řádcích a zasílány jako textové řetězce do příslušného mapperu. Mapper tento řádek přečte ze standardního vstupu a předá jako textový řetězec do instance třídy `FlowMapper` implementující fázi mapping. Tato třída provede deserializaci textového řetězce

ve formátu JSON do příslušného objektu a ten předává do metody `Map`. Výstupem zpracování metody `Map` je opět objekt, který se před odeslání na standardní výstup opět serializuje do formátu JSON. Tato serializac se provádí v kontextu zpracování fáze mapping, do kterého se jednotlivé výstupy zapisují. Po převodu do formátu JSON se opět pracuje daty, jako s textovým řetězcem. Jeden výstup z fáze mapping je tedy textový řetězec s následující strukturou.

```
<klíč>\t{"Payload":<data zprávy>, "Captured":<čas zachycení>}
```

Kde položka `<klíč>` je pětice hodnot identifikující datový tok uvedená v kapitole 5.4.2, který se přenáší ve formátu.

```
<zdrojová IP>;<cílová IP>;<zdrojový port>;<cílový port>;<protokol>
```

Hodnoty označené klíčem vstupují do další fáze zpracování zvané shuffling. Funkčnost této fáze je zajištěna v implementaci Apache Hadoop frameworku a lze ji konfigurovat, jakým způsobem se bude klíč porovnávat. Jejím vstupem jsou hodnoty ve formátu, v jakém byly zapsány na výstup ve fázi mapping. Jsou zde ovšem shlukovány podle hodnoty klíče, takže na počátku fáze reducing musí reducer přečíst ze standardního vstupu vždy všechny hodnoty se stejným klíčem a předat je do metody `Reduce` třídy `FlowReducer`. Jelikož je metoda `Reduce` v rodičovské třídě označena modifikátorem `sealed`, tak se zabrání přepsání této metody ve zděděné třídě. V této metodě se provede deserializace z formátu JSON do příslušného objektu a výsledek se předá jiné metody Reduce (s jinými parametry). V ní se poté provede detekce protokolu a zpracování všech zpráv. Jejím výstupem jsou data zapisovaná na jeden řádek na standardní výstup se strukturou.

```
<klíč>\t
{
    "Header":{
        "SourceIP":<zdrojová IP adresa>,
        "DestIP":<cílová IP adresa>
        "SourcePort":<zdrojový port>
        "DestPort":<cílový port>
        "TransProtocol":<transportní protokoů>
    },
    "StartTime":<datum a čas první zprávy>,
    "StopTime":<datum a čas poslední zprávy>,
    "Messages": [
        {
            "Name":<název protokolu zprávy>,
            "CaptureDataTime":<datum a čas zachycení zprávy>,
            "Fields": {
                <klíč položky>:<hodnota položky>
            }
        }
    ],
}
```

Ze způsobu příjmu dat se může zdát, že se fáze reducing provádí sekvenčně. Ovšem je nutné vzít do úvahu, že počet spuštěným reducerů řídí Apache Hadoop framework. Z nich každý reducer dostane část dat z výstupu fáze shuffling a nad nimi pracuje sekvenčně.

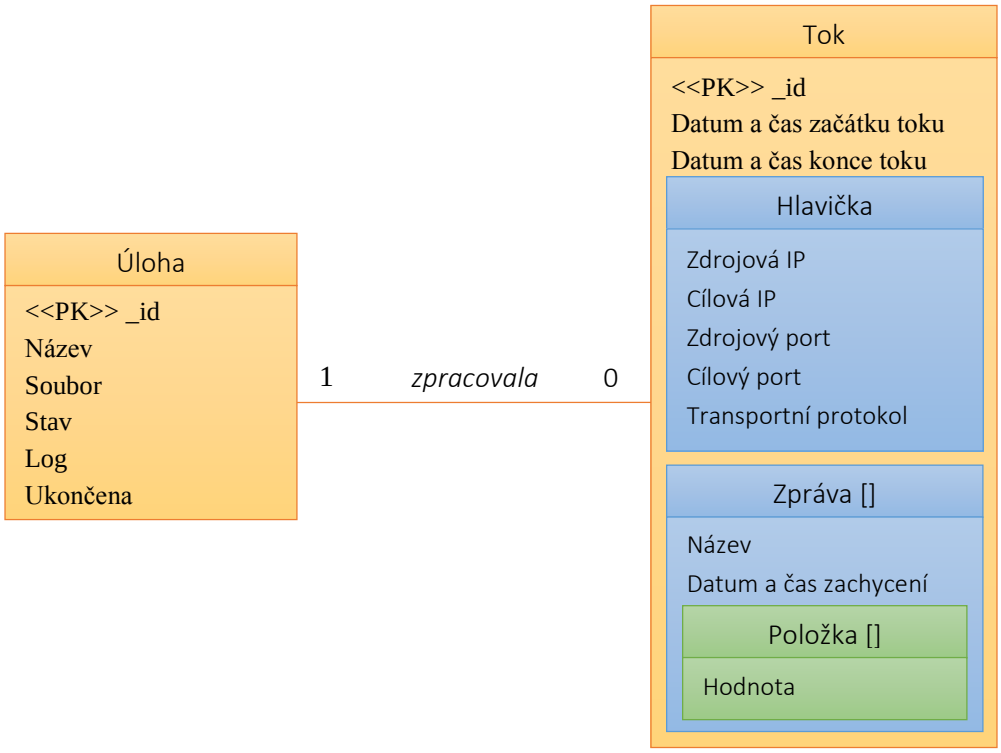
5.4.5 Ukládání výsledků do databáze MongoDB

Způsob zpracování výstupu z MapReduce úlohy určuje konkrétní implementace výstupního formátu. Výstupní formát je reprezentován třídou v jazyce Java stejně jako v případě vstupního formátu. Musí ale implementovat rozhraní `org.apache.hadoop.mapred.OutputFormat<K,V>`, které je jiné než v případě vstupního formátu.

Ve výsledné aplikaci potřebujeme ukládat výstup MapReduce úlohy do databáze MongoDB. Využíváme přitom pluginu MongoDB Connector For Hadoop, který obsahuje implementaci výstupního formátu pro ukládání výsledků MapReduce úloh do databáze MongoDB. Ten obsahuje implementaci výstupního formátu ve třídě `MongoOutputFormat<K,V>` umístěné ve jmenném prostoru `com.mongodb.hadoop.mapred`, kterou využíváme jako rodičovskou třídu. Musíme totiž implementovat vlastní zapisovač tzv. writer do MongoDB, protože k výslednému síťovému toku, který získáme z fáze reducing, potřebujeme dodat identifikátor úlohy. Tento identifikátor zadáváme do parametru pro Hadoop streaming, jak je uvedeno v kapitole .5.3.1 Získáme tím spojení výsledných síťových toku k dané úloze.

5.5 Struktura databáze

Zpracovaná data síťové komunikace se na konci zpracování MapReduce úlohy ukládají přes výstupní formát napsaný v jazyce Java do databáze MongoDB. Dále se v aplikaci tato databáze využívá pro ukládání vyvářených úloh, které se spouští na Apache Hadoop Clusteru. Výsledná databáze tedy obsahuje pouze dvě kolekce. Jednu pro ukládání informací o prováděných MapReduce úlohách a druhá pro ukládání výstupu zpracování MapReduce úloh. Jediné, co se do databáze neukládá je nastavení aplikace. To je řešeno přes XML soubor, který se nachází ve složce uživatelského profilu.



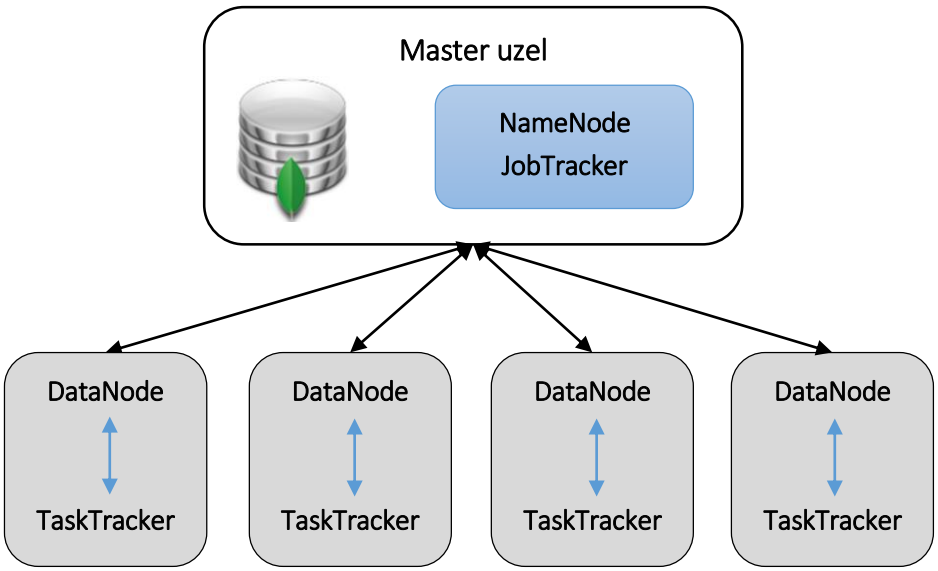
Obrázek 5.4: Struktura databáze v aplikaci.

Návrh této databáze se prováděl souběžně s návrhem entit, jejíž data se do ní budou ukládat. Entity totiž představují objekty pro ukládání a načítání dat z databáze. To je řešeno pomocí serializace daného objektu při ukládání do databáze a při načítání dat z databáze se provádí inverzní operace k serializaci a to deserializace, které převede data z formátu uloženém v databázi do instance objektu. V aplikaci se tedy vždy pracuje s objekty, které reprezentují dané položky z databáze. To je zajištěno díky zmíněné serializaci a deserializaci, která je implementována v ovladači databáze MongoDB pro jazyk C#. Použité schéma databáze, se kterým se v aplikaci pracuje, zobrazuje obrázek Obrázek *5.4*.

6 Zhodnocení dosažených výsledků

6.1 Testovací prostředí

Pro účely testování byl ve virtuálním prostředí vytvořen Apache Hadoop cluster. Tento cluster se skládal celkem z pěti uzlů. Hlavní uzel tzv. master, jak je zobrazeno na obrázku Obrázek *6.1*, zahrnoval úložiště metadat pro souborový systém HDFS (služba NameNode), službu pro řízení MapReduce úloh (služba JobTracker) a v neposlední řadě také databázi MongoDB pro ukládání výsledků zpracování. Čtyři podřízené uzly tzv. slave uzly sloužili jako úložiště v souborovém systému HDFS (služba DataNode), a pak také vykonávali části úloh přijaté od master uzlu (služba TaskTracker).



Obrázek 6.1: Konfigurace testovacího clusteru.

Testovací prostředí bylo vytvořeno v nástroji VMware ESXi, který slouží pro správu virtuálních zařízení nad fyzickým zařízením. Na každém virtuálním počítači byl nainstalován operační systém Windows Server 2012. Framework Apache Hadoop byl použit z distribuce Hortonworks Data Platform for Windows ve verzi 1.3, který obsahuje nástroj Apache Hadoop ve verzi 1.2.

6.2 Způsob testování

Testování probíhalo se třemi PCAP soubory o různých velikostech. Cílem této volby bylo zjistit chování s ohledem na velikost souboru. Soubory byly vytvořeny, tak aby obsahovali stejnou zachycenou komunikaci. První soubor o velikosti 176 MB byl podmnožinou druhého souboru od velikosti 297 MB. Tento druhý soubor se poté duplikoval a spojil v jeden soubor, čímž vznikl třetí testovací soubor o dvojnásobné velikosti oproti druhému. Tímto způsobem zvolení testovacích souborů bylo dosaženo toho, že parsery aplikačních protokolů budou zpracovávat stejná dat a nebudou ovlivňovat dobu zpracování. Jinak řečeno pokud aplikační parser bude sekvenčně zpracovávat dvakrát stejná data, tak doba provádění by měla být dvojnásobná. Díky tomu dostaneme na výstupu porovnatelné výsledky, které nebudou ovlivnění různou dobou zpracování různých protokolů.

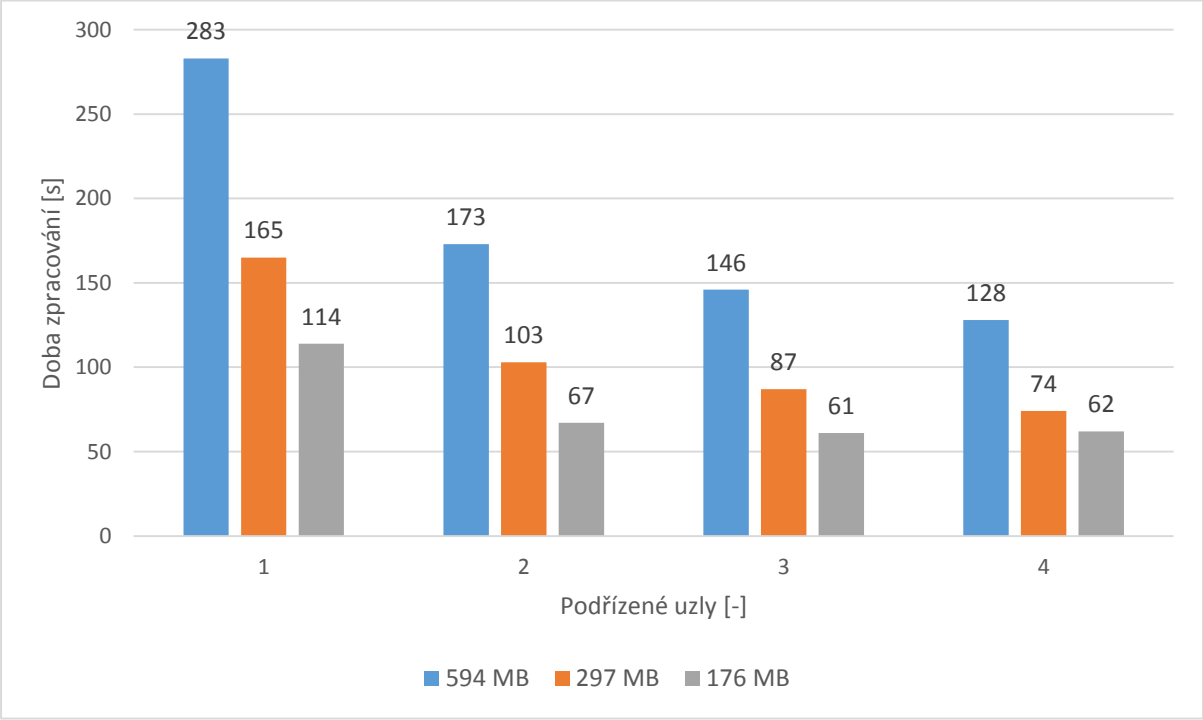
Pro každý soubor ze tří testovacích se provádělo několik měření, abychom vyloučili vliv rozptylu mezi dobami zpracování v jednotlivých měřeních. Toto měření se provádělo vždy na konfiguraci

clusteru od čtyř podřízených uzlů až do jednoho podřízeného uzlu. Přitom po každé sadě měření se odebral jeden podřízený uzel. Každé z měření se provádělo na se čtyřmi *reduce* úlohami, jejichž počet je nutné explicitně specifikovat pomocí definice v parametru `mapred.reduce.task`. Pokud není tento parametr v Hadoop streaming uveden, tak se spustí pouze jedna reduce úloha. Stejný způsobem je možné definovat také počet *map* úloh. Jedná se o parametr `mapred.map.task`, který nebyl nastavován a byl ponechán do výchozího nastavení. Při němž se fáze *mapping* rozdělí do tolika map úloh, z kolika bloků v HDFS se skládá vstupní soubor.

Výsledná doba zpracování úlohy se neměřila přímo v aplikaci. Tato hodnota se pro každou úlohu získala z komponenty JobTracker běžící na master uzlu, kde jsou dostupné informace o jednotlivých úlohách. Volba na tento způsob měření vycházela z použití vzdáleného zpracování, které má rychlejší odezvu při práci se souborovým systémem HDFS. Při tomto zpracování se na ukončení dotazujeme serveru v určitých časových interval, čímž by bylo měření nepřesné

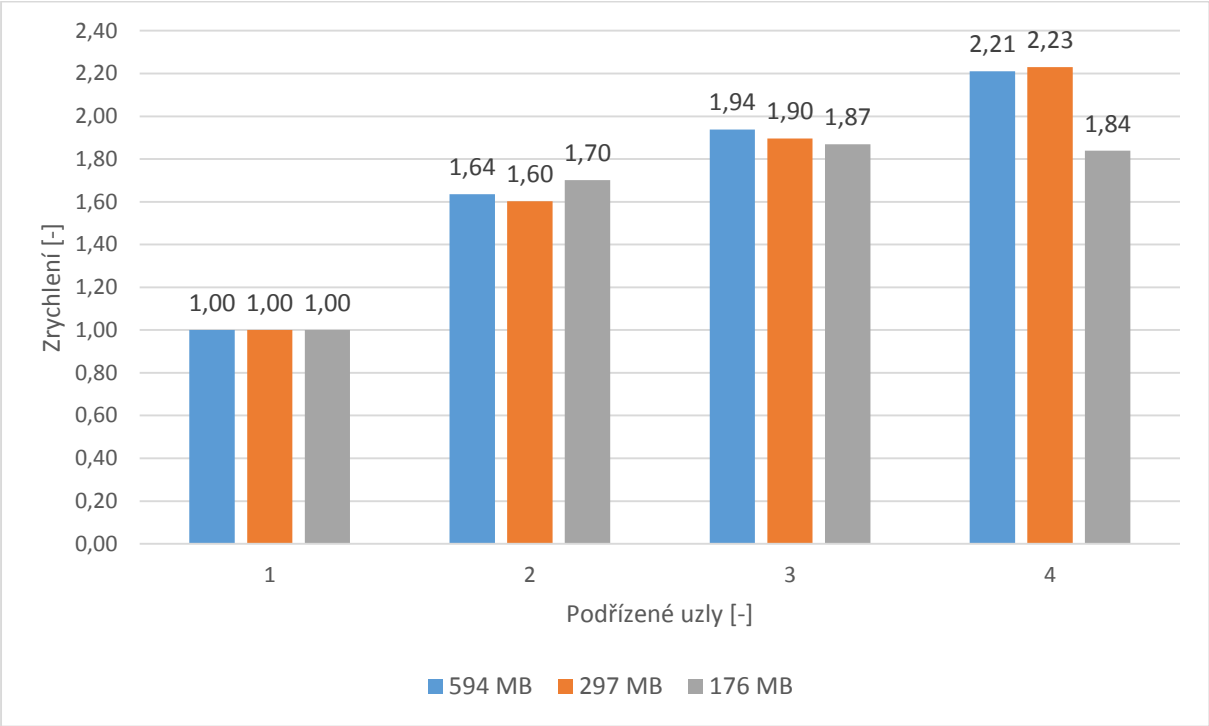
6.3 Dosažené výsledky

Naměřené výsledky byly zpracovány do dvou grafů, z nichž lze vyčíst informace o chování výsledného řešení v testovacím prostředí. První graf Graf *6.1* zobrazuje absolutní dobu zpracování testovacích souborů. Absolutní hodnoty v tomto grafu nejsou hlavním zdrojem informací. Lepší je se zaměřit na rozdíly při zpracování na různém počtu uzlů. Poté je z grafu patrné, že největší úsporu v době zpracování docílíme zvýšením počtu podřízených uzlů, které zpracovávají jednotlivé úlohy *map* a *reduce*, z jednoho na dva uzly. Přidáváním dalších zpracovávajících uzlů, dosáhneme také rychlejšího zpracování, které ovšem roste o polovinu pomaleji než při zvýšení z jednoho zpracujícího uzlu na dva. Důležité ovšem je, že poté doba zpracování klesá lineárně. To ale neplatí pro nejmenší testovací soubor o velikosti 176 MB. Zde je patrné zrychlení zpracování přidáním druhého uzlu, ale přidáním dalších uzlů nedosáhneme dalšího zrychlení. To je dáno velikostí souboru, protože se nevyužije naplno celá dostupná výpočetní kapacita, když se zpracování rozděluje po blocích, v jakých jsou části souboru uloženy v HDFS.



Graf 6.1: Doba zpracování testovacích souborů.

Zrychlení pří různém počtu podřízených uzlů zobrazené na grafu Graf *6.2* vychází z dob provádění MapReduce úloh zobrazených na grafu Graf *6.1*. Jedná se tedy o jiný pohled na data. Udává přitom, kolikrát se zrychlí doba provádění úlohy při daném počtu podřízených uzlů v Apache Hadoop clusteru v porovnání s jedním prováděcím uzlem. Z grafu lze zjistit, že přidáním druhého uzlu se doba zpracování zrychlí v průměru 1,65krát. Přidáváním dalšího uzlu roste zrychlení s každým přidaným uzlem o hodnotu přibližně 0,3. Dále je patrném, že při zpracování třetího souboru o velikosti 176 MB se při třech a čtyřech zpracovávajících uzlech nedokáže plně využít dostupná výpočetní kapacita. Maximální zrychlení jaké jsme při zpracování tohoto souboru schopni dosáhnout je přibližně 1,85krát. Je to dáno malou velikostí zpracovávaného souboru.



Graf 6.2: Zrychlení při různém počtu podřízených uzlů.

7 Srovnání s dostupnými nástroji a možná rozšíření aplikace

Tato kapitola se zaměřuje na zhodnocení výsledné aplikace oproti jiným nástrojům pro analýzu síťové komunikace. Srovnání se zaměřuje spíše na porovnání funkcí, které nabízí či nenabízí oproti jiným řešením. Druhá část kapitoly je věnována rozvaze o dalším možném rozšíření aplikace s nastíněním jakým směrem by se mohlo jít v dalším vývoji.

7.1 Srovnání s dostupnými nástroji

Implementovaná aplikace zahrnuje dva základní funkční celky. Prvním z nich představuje zpracování dat ze síťové komunikace, kde se hledají síťové toky a následně se zpracovávají jednotlivé zprávy patřící do daného toku. Druhá část aplikace poté umožňuje provádět analýzu nad výstupními daty ze zpracování síťové komunikace. Na základě toho rozdělení byl vybrán systém Netflow a program Winreshark. Se systémem Netflow se bude porovnávat první část aplikace s ohledem na detekci síťových toků a jejich celkového zpracování. Možnosti analýzy dostupné ve výsledné aplikaci jsou srovnány právě s možnostmi v programu Wireshark, který je jedním ze zástupců paketových analyzátorů.

7.1.1 Porovnání se systémem NetFlow

Při srovnání se systémem Netflow se zaměříme především na způsob detekce síťových a dat, které jsou o něm zaznamenávány. V implementované aplikaci identifikujeme síťový tok podle pětice hodnot obsahující zdrojovou a cílovou IP adresu, zdrojový a cílový port a protokol na transportní vrstvě. Systém Netflow k této pětici přidává ještě dvojici hodnot zahrnující název logického rozhraní a typ služby (pole ToS z IP verze 4). Z toho vyplívá, že dělení na síťové toky se v systému Netflow provádí jemněji než ve výsledné aplikaci, kde se používá o dvě hodnoty méně. Vy výsledku tedy identifikuje více síťových toků. K tomu také přispívá, že sběr statistik k danému toku se provádí jen tak dlouho, pokud nenastane exspirace. Ve výsledné aplikaci ovšem žádná exspirace pro ukončení daného toku není brána v úvahu. Je to dáno jednak zpracováním přes programovací model MapReduce a pak také, že zpracováváme komunikaci, která již proběhla. Během zpracování se v programovací modelu MapReduce se provádí ve fázi mapping detekce síťového. Tato fáze se v případě využití více než jedno uzlového clusteru provádí na více uzlech a tedy paralelně. Detekce exspirace např. kontrolou příznaků RST a FIN v hlavičce TCP paketu by byla úspěšná pouze v jednom případě a ve výsledku by došlo ke smíchání paketů k jiným toků. Naopak v systému Netflow se musí být exspirace síťové toku používat, protože tento systém analyzuje právě proudící data a nemá informaci o tom, jak dlouho bude daný síťový tok aktivní. To je v případě zpracovávání již proběhlé zachycené komunikace omezeno velikostí zpracovávaného souboru.

Pokud budeme srovnávat výstupy, které jednotlivé systémy produkují, tak zjistíme, že oba jsou zaměřeny na jinou oblast. V případě Netflow je zaměření na informace o vlastní síťovém přenosu. Na druhé straně implementovaná aplikace provádí parsování aplikačních protokolů. Snaží se tedy získat informace o tom, co se síťovou komunikací přenášelo.

7.1.2 Porovnání s programem Wireshark

Program Wireshark zastupuje skupinu programů, které umí zachytávat síťovou komunikaci na zvoleném síťovém rozhraní případně ji načíst z uloženého souboru a poté ji analyzovat. Pracuje způsobem okamžitého zpracování odchycené komunikace, která může být ihned analyzována. Z toho vyplívá, že tento program neprovádí shlukování komunikace do síťových toků, ale přímo zpracovává datové rámce v pořadí, jak se vyskytly na síťovém rozhraní. Na základě toho se zaměříme spíše na srovnání funkcí pro analýzu zachycené komunikace.

Analýza je v programu Wireshark založena na protokolových parserech, kterých implementuje velké množství. Pomocí nich zpracovává datové jednotky na jednotlivých vrstvách provozu. Zato ve výsledné aplikaci jsme zaměřeni na získávání dat z aplikačních protokolů. Získáváme, ale pouze některé hodnoty, ale to záleží na konkrétní implementaci parseru. Na druhé straně program Wireshark zpracuje celý síťový rámec takovým způsobem, že poté můžeme určit, která data v poli bajtů odpovídají kterým zpracovaným položkám jednotlivých datových jednotek. To je výborné zejména při ladění síťové komunikace u vyvíjené aplikace. Dobře se tím provádí kontrola, co se přenáší. K tomuto účelu, ale není výsledná aplikace určena. Apache Hadoop framework zahrnuje totiž vždy režii spojenou s vytvořením a správou úlohy. Tato režie se víc promítne při zpracování menších souborů, protože ji nelze více omezit. Výsledná aplikace je určena zejména pro zpracování většího množství dat, přičemž se pozitivně promítne použití programovacího modelu MapReduce.

7.2 Možná rozšíření aplikace

Z hlediska dalšího možné vývoje aplikace a přidávání dalších funkcí je velký potenciál k rozšíření. Řekl bych, že „základní“ funkčnost aplikace zajišťuje. Jako rozšíření dále bych viděl možnost zpracování více vstupních formátu a implementaci většího počtu protokolových parserů.

Největší slabinu vidím v možnostech analýzy. Bylo by dobré implantovat funkce, které by umožnili analyzovat data získaná z parseru protokolu. Ty jsou totiž uložena v databázi ve formě klíč hodnota. K tomu je nutné znát od příslušného parseru jaké hodnoty zpracovává. Jako příklad takovéto analýzy mohu uvést třeba počet výskytů jednotlivých požadavků v protokolu HTTP s přihlédnutí na cílovou doménu nebo například získávat statistiky o odesílatelích v e-mailových zprávách a zjišťovat tímto způsobem možné zdroje nevyžádané pošty. Z tohoto ohledu poskytuje aplikace analýzu z hlediska sítových toků, ale neumí právě zmíněné pokročilejší analýzy pracujících z daty získaných z aplikačních protokolů.

Druhou částí pro zlepšení vidím v malém počtu zpracovávaných protokolů. V aplikaci by bylo nejlepší, aby se daly přidávat parsery protokolů formou pluginů. To ovšem není z hlediska použití Hadoop streaming dlouhodobě udržitelné, protože s přibývajícími pluginy by narůstala délka příkaz spouštěcího MapReduce zpracování. Přidání dalších parserů protokolů tedy vyžaduje rekompilaci knihovny `NetfloxMontana.Parse.dll`. V tom ovšem nevidím nějaký problém. V případě, že by se zvýšil počet zpracovávaných protokolů, tak se stane aplikace zajímavější pro širší využití.

Z hlediska zvýšení počtu zpracovávaných vstupní formátů podporuje aplikace jen formát PCAP soubor jehož struktura byla popsána v kapitole 2.2.1.2. Existuje více druhů PCAP formátů a také jsou další formáty ukládání zachycené síťové komunikace. Kromě rozšíření o tyto formáty by mohlo být implementováno zachytávání síťové komunikace ze síťového rozhraní a následné zpracování takto získaných dat na Apache Hadoop clusteru. To by mohlo fungovat způsobem, že by se odchytnulo určité množství síťové komunikace nebo by se odchytávalo po určitý čas příp. kombinací obou přístupů. Zachycená data by se přímo převáděla na vstupní data ke zpracování a byla by ukládána do souboru.

Až by se dosáhlo některého z omezení, tak by se zachycená data ukládala do jiného souboru. Pro předešlí soubor by se vytvořila úloha a spustilo se zpracování. Takto by mohla být analyzována veškerá data procházející některým uzlem sítě. Tento uzel by ale musel umožňovat běh aplikací postavených na .NET frameworku. Tím by se ovšem při vysokém zatížení sítě toto zatížení zvyšovalo, když by se vstupní data úloh přenášela na zpracující cluster. Pokud by byl uzel, na němž by aplikace běžela zároveň master uzlem Apache Hadoop clusteru nebo jiným uzlem v daném clusteru, tak by se tento problém eliminoval.

8 Závěr

Impulsem pro vytvoření této práce byla myšlenka zpracovávat zachycenou síťovou komunikaci paralelně v distribuovaném prostředí a výsledky zpracování využít k analýze provozu na síti. Důvodem pro využití distribuovaného výpočtu je fakt, že síťová komunikace představuje obrovské množství dat, které je potřeba zpracovat. Distribucí máme možnost zpracovat takové velké množství data za přijatelný čas.

Účelem práce bylo prozkoumat a ověřit možnosti distribuovaného zpracování síťových dat. Přičemž za distribuované prostředí byl využit cluster počítačů vyžívající technologii Apache Hadoop. Z těchto požadavků vychází i struktura práce a její jednotlivé kapitoly.

Na počátku bylo nejprve nutné prozkoumat, jaká dat budou na vstupu zpracovávána a jejich možnosti paralelního zpracování s využití programovacího modelu MapReduce. Nakonec dle zjištěných skutečností bylo do návrhu zakomponováno, že zachycená síťová data se budou zpracovávat po jednotlivých paketech. Za vstupní formát byl zvolen rozšíření formát PCAP. Data v tomto formátu nelze, ale přímo použít, jak vstup při MapReduce zpracování, protože při rozdělování souboru se nejsme schopni synchronizovat na začátek logického bloku. Proto se data před zpracováním převádí do formátu JSON, kde máme informace o jednom paketu na jednom řádku a se vstupními daty pracujeme jako s textem po řádcích. Poté již není nutné vstup zpracovávat sekvenčním způsobem a síťové toky ve fázi *mapping* můžeme detekovat paralelně v několika *Map* úlohách. Díky určení síťových toků máme ve fázi *reducing* seskupeny vždy všechny pakety označené další klíčem a můžeme dále zpracovávat jejich obsah.

Navržené řešení bylo následně implementování v jazyce C# jako Windows Presentation Foundation aplikace a několik DLL knihoven. Při implementaci MapReduce zpracování byl využit balík Microsoft .NET SDK For Hadoop. Tento balík neobsahuje kompletní implementaci konfigurace streamovaných úloh s použitím Hadoop streaming. Z tohoto důvodu bylo nutné implementovat vlastní řízení zpracování streamovaných úloh, které rozšiřuje implementaci v uvedeném balíku.

Jakmile bylo výsledné řešení implementováno, tak byly provedeny testy s různými velikostmi zpracovávaných vstupních souborů. Tyto testy byly zaměřeny na zhodnocení vlivu distribuovaného prostředí při použití různého počtu pracujících uzlů. Z jejich výsledků se potvrdila škálovatelnost doby zpracování při použití frameworku Apache Hadoop. Velice důležitým poznatkem z testů bylo, že velikost vstupního souboru ovlivňuje dosažené zrychlení oproti zpracování na jednomu uzlu. Pokud totiž není vstupní soubor dostatečně velký, tak se nevyužije dostupná výpočetní kapacita clusteru a zrychlení se od určitého počtu zpracujících uzlů dále nezvyšuje. Můžeme tedy říci, že velikost zrychlení roste lineárně s velikostí zpracovávaného souboru, pokud máme dostupnou výpočetní kapacitu.

Celkově si z této práce odnáším mnoho zkušeností. Zejména musí zmínit framework Apache Hadoop, který mi přidal díky programovacímu modelu MapReduce, nový pohled na možnost zpracování dat, s nímž jsem se doposud nesetkal. Líbí se mi na tomto přístupu jeho elegance, se kterou se v něm úlohy řeší. Řekl bych, že tento přístup vystihuje, když se řekne: „V jednoduchosti je síla“. Díky relativně jednoduchým činnostem v metodách Map a Reduce dosáhneme požadovaného výsledku.

Literatura

[1] KOHLER, , M. HANDLEY a S. FLOYD. In: *RFC4340* [online]. 2006 [cit. 2014-Leden-01]. Dostupné z: http://www.rfc-editor.org/rfc/rfc4340.txt

[2] DOSTÁLEK, L. a A. KABELOVÁ. *Velký průvodce protokoly TCP/IP a systémem DNS*. Praha: Computer Press, 2000. ISBN 80-7226-323-4.

[3] OREBAUGH, A. et al. *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Rockland: Syngress Publishing, 2007. ISBN-13: 978-1-59749-073-3.

[4] HARRIS, G. Internet Assigned Numbers Authority. In: *MIME type: application/vnd.tcpdump.pcap* [online]. 30. 03. 2011 [cit. 2014-Leden-02]. Dostupné z: http://www.iana.org/assignments/media-types/application/vnd.tcpdump.pcap

[5] The Wireshark Wiki. *Development/LibpcapFileFormat* [online]. 29. Červenec. 2013 [cit. 2014-Leden-02]. Dostupné z: http://wiki.wireshark.org/Development/LibpcapFileFormat

[6] Cisco Systems. *NetFlow Version 9 Flow-Record Format* [online]. 2011 [cit. 2014-Leden-06]. Dostupné z: http://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html

[7] SystemOnLine. *Big data* [online]. 2011 [cit. 2014-Leden-03]. Dostupné z: http://www.systemonline.cz/clanky/big-data.htm

[8] VENNER, J. *Pro Hadoop*. Apress, 2009. ISBN: 978-1-4302-1942-2.

[9] LAM, C. *Hadoop in Action*. Stamford: Manning Publications, 2011. ISBN: 9781935182191.

[10] SQL SERVER TEAM. SQL Server Team Blog. In: *Microsoft's Big Data Roadmap & Approach* [online]. 13. 10. 2011 [cit. 2014-05-14]. Dostupné z: http://blogs.technet.com/b/dataplatforminsider/archive/2011/10/13/microsoft-s-big-data-roadmap-amp-approach.aspx

[11] Microsoft TechNet. *HDInsight Server* [online]. 2014 [cit. 2014-Leden-06]. Dostupné z: http://technet.microsoft.com/en-us/library/dn247618.aspx

[12] MICROSOFT. Introduction to WPF. *Microsoft Developer Network* [online]. 2014 [cit. 2014-05-17]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/aa970268(v=vs.110).aspx

[13] NATHAN, A. *WPF 4 UNLEASHED*. Pearson Education, 2010. ISBN 978-0-672-33119-0.

[14] MongoDB. *Partners* [online]. 2014 [cit. 2014-05-14]. Dostupné z: http://www.mongodb.com/partners

[15] MONGODB, I. MongoDB Manual 2.6.1. *Sharding* [online]. 2014 [cit. 2014-05-14]. Dostupné z: http://docs.mongodb.org/manual/sharding/

[16] O'BRIEN, M. MongoDB. *MongoDB Blog* [online]. 2013 [cit. 2014-05-16]. Dostupné z: http://blog.mongodb.org/post/57611443904/mongodb-connector-for-hadoop

[17] Mirosoft Developer Network. *Implementing the MVVM Pattern Using the Prism Library 5.0 for WPF* [online]. 2014 [cit. 214-05-09]. Dostupné z: http://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx

[18] INC, H. Apache HCatalog. *Hortonworks* [online]. 2011-2014 [cit. 2014-05-16]. Dostupné z: http://hortonworks.com/hadoop/hcatalog/

[19] In: *RFC 793 - Transmission Control Protocol* [online]. 1981 [cit. 31-12-2013]. Dostupné z: http://tools.ietf.org/html/rfc793

[20] In: *RFC 791 - INTERNET PROTOCOL* [online]. 1981 [cit. 2013-12-30]. Dostupné z: http://tools.ietf.org/html/rfc791

[21] In: *Windows Azure HDinsight Datasheet* [online]. 2014 [cit. 2014-Leden-06]. http://download.microsoft.com/download/D/6/1/D6105D2F-7D6E-4875-AF91-A3D7262ADB17/Windows_Azure_HDInsight_Datasheet.pdf